Parallel Algorithms for Top-*k* Query Processing

Neil Zhenqiang Gong[1]
EECS Department
UC Berkeley
Berkeley, CA
neilz.gong@berkeley.edu

Guang-Zhong Sun
School of Computer Science and Technology
University of Science and Technology of China
Hefei, P. R. China
gzsun@ustc.edu.cn

Dongqu Chen
School of Computer Science and Technology
University of Science and Technology of China
Hefei, P. R. China
cdq2012@mail.ustc.edu.cn

Abstract—**The general problem of answering top-*k* queries can be modeled using lists of objects sorted by their local scores. Fagin et al. proposed the "middleware cost" for a top-*k* query algorithm, and proposed the efficient sequential Threshold Algorithm (TA). However, since the size of the dataset can be incredible huge, the middleware cost of sequential TA may be intolerable. So, in this paper, we propose parallel algorithms to process top-*k* queries and analyze their middleware costs. Intuitively, a naive parallel algorithm, called PTA (parallel-TA), evenly partitions the original dataset into P (the number of processors) subdatasets. Each processor finds top-*k* results of one corresponding subdataset using TA algorithm. Then these results are merged to get the final top-*k* answers. Motivated by the idea of partitioning objects, we take a further step to partition D into n subdatasets according to their *degree of domination*. Based on this partition, we propose EPTA (Enhanced-PTA) algorithm. Under PRAM-CRCW model, the middleware cost of PTA is $O(nm^2/P)$ while the average middleware cost of EPTA is $O(km^2(\ln n)^{m-1}/(m-1)!)$ under the assumption that scores in different lists are independently distributed, where *n* is the dataset size and *m* is the number of lists. Extensive experiments show that the speedup ratios of EPTA are significantly higher than those of PTA.**

Keywords: Top-k query; TA; Parallel top-k algorithm;

I. Introduction

Assume there are *n* objects and every object has m attributes, for each attribute the object has a local score. These local scores can be aggregated to an overall score by an aggregate function g, and we want to know which *k* objects have the largest overall scores. This scenario is generalized as "top-*k* queries". For example, there are many kinds of cell phones in the market. Each cell phone may have price and standby time attributes. Each attribute has a numeric score ranging in the interval [0, 1], where the score of price attribute is reverse proportional to price while the score of standby time attribute is proportional to standby time. Bob prefers a cheap cell phone, and does not care much about the standby time. So he may issue a top-*k* query with aggregate function as $g = 0.8 \times price + 0.2 \times standby\ time$. On the other hand, Alice prefers a longer standby time cell phone rather than a cheaper one. So she may issue a top-*k* query with aggregate function as $g = 0.2 \times price + 0.8 \times standby\ time$.

Top-*k* queries have attracted considerable attention because of its wide use in many areas such as P2P systems and sensor networks [4][5], network and system monitoring [6][7], information retrieval [8][9][13], spatial data analysis [14][15], etc. The main reason for such attention is that top-*k* queries avoid overwhelming the user with large numbers of uninteresting answers which are resource-consuming.

A general and simple model proposed by Fagin et al.[16] is that the dataset consists of m sorted lists with n data items. Each data item can be accessed through *sorted access* or *random access*. Sorted access iteratively reads data items sequentially, whereas a random access is a request for a data item in some list given the object's ID. The middleware cost of a top-*k* query algorithm is $a_s c_s + a_r c_r$, where a_s is the number of sorted accesses performed, a_r is the number of random accesses performed, c_s is the cost per sorted access, and c_r is the cost per random access. Under this model, Fagin et al.[16] proposed the efficient sequential Threshold Algorithm (TA).

However, since the size of the dataset can be incredible huge, the middleware cost of sequential TA may be intolerable. So, in this paper, we propose parallel algorithms to process top-*k* queries. A naïve parallel algorithm, called PTA (parallel-TA), evenly partitions the original dataset into P (the number of processors) subdatasets. Each processor finds top-*k* results of one corresponding subdataset using TA algorithm. Then these results are merged to get the final top-*k* answers. The middleware cost of PTA is $O(nm^2/P)$ under PRAM-CRCW model, where *n* is the dataset size and *m* is the number of lists. Motivated by the idea of partitioning the dataset, we take a further step to partition the original dataset into n subdatasets (some subdatasets may be empty) according to their *degree of domination* (see definition 2). Based on this partition, we propose EPTA (Enhanced-PTA) algorithm, which works in three phases. In the first phase, EPTA distributes *k* subdatasets onto the P processors using a heuristic distribution algorithm. In the second phase, the ith processor p_i uses a new algorithm designed by us to compute top-($k - d_i$) results among the subdatasets distributed onto it, where d_i is the minimum degree of domination of the objects distributed onto p_i. In the third phase, EPTA combines the results returned in the second phase to get the final top-*k* answers. We estimate the cardinality of each subdataset as $O((\ln n)^{m-1}/(m-1)!)$ under the assumption that the scores in different lists are independent, based on which, we deduce the average middleware cost of EPTA to be $O(km^2(\ln n)^{m-1}/(m-1)!)$ under PRAM-CRCW model.

[1]This work was completed when Neil Gong was undergraduate student of USTC
[2]Corresponding author

Furthermore, we do extensive experiments to evaluate our algorithms. The experimental results show that the speedup ratios of EPTA are significantly higher than those of PTA.

In this paper, our contributions are as follows:

- We propose two parallel algorithms for top-*k* queries processing, i.e. parallel-TA (PTA) algorithm and Enhanced-PTA (EPTA) algorithm. To the best of our knowledge, this is the first paper proposing parallel algorithms to process top-*k* queries.
- We analyze the middleware cost of both PTA and EPTA under PRAM-CRCW model. Specifically, the middleware cost of PTA is $O(nm^2/P)$. And the average middleware cost of EPTA is $O(km^2(\ln n)^{m-1}/(m-1)!)$ in the cases where the local scores in different lists are independently distributed.
- We do extensive experiments to evaluate our algorithms. The experimental results show that the speedup ratios of EPTA are significantly higher than those of PTA.

The rest of this paper is organized as follows. In section II, we formally define the problem and review TA algorithm. In section III, we describe our algorithm PTA. Thereafter, we show our algorithm EPTA and analyze its complexity in section IV. In section V, we show the experimental results. Finally, in section VI, we conclude this paper and introduce our future work.

II. Problem Definition and TA Algorithm

In this section, we formally define top-*k* problem and review the sequential TA algorithm.

L_1	L_2	L_3
(T_2,0.9)	(T_1,0.8)	(T_3,0.7)
(T_1,0.5)	(T_4,0.7)	(T_4,0.6)
(T_3,0.4)	(T_2,0.6)	(T_2,0.5)
(T_5,0.3)	(T_3,0.3)	(T_5,0.2)
(T_4,0.1)	(T_5,0.2)	(T_1,0.1)

Figure 1. Three sorted lists

Our model of the dataset (see Fig. 1) can be described as follows [16]: assume the dataset D consists of m sorted lists, which are denoted as $L_1, L_2, \cdots, L_m$. Each sorted list consists of n data items. Each data item is a pair $(t, s_i(t))$, where t is an object, $s_i(t)$ is t's ith local score which is a real number in the interval [0, 1]. Sorted list means that objects in each list are sorted in descending orders according to their local scores. Each data item can be accessed through *sorted access* and *random access*. Sorted access iteratively reads data items sequentially, whereas a random access is a request for a data item in some list given the object's ID. Fagin et al. introduced middleware cost for top-*k* query algorithms. The middleware cost of an algorithm is $a_s c_s + a_r c_r$, where a_s is the number of sorted accesses performed, a_r is the number of random accesses performed, c_s is the cost of a single sorted access, and c_r is the cost of a single random access.

For a top-*k* query with a user-defined positive integer *k* and a *m*-dimensional aggregate function g, the result set Y is a set of objects such that $Y \subseteq D$, $|Y| = k$ and $\forall t_1, t_2$: $t_1 \in Y$, $t_2 \in D - Y$, it holds that $g(t_1) \geq g(t_2)$. In this paper, we assume the aggregate function g to be increasingly monotonic. An *m*-dimensional function g is increasingly monotonic if $g(x_1, x_2, \cdots, x_m) \leq g(x_1^{'}, x_2^{'}, \cdots, x_m^{'})$, whenever $x_i \leq x_i^{'}$ for every i. Many popular aggregate functions, e.g. Min, Max, Sum, are increasingly monotonic.

Fagin et al. [16] proposed the efficient sequential Threshold Algorithm (TA) to process top-*k* queries with any increasingly monotonic function. Its middleware cost is $O(nm(c_s + (m-1)c_r))$. Since $\min\{c_s, c_r\} \times m \leq c_s + (m-1)c_r \leq \max\{c_s, c_r\} \times m$, we have $O(nm(c_s + (m-1)c_r)) = O(nm^2)$. So the middleware cost of TA is $O(nm^2)$. TA is described in Fig. 3. In the description, we replace the notations with ours.

Threshold Algorithm (TA)

```
1. Do sorted access in a round-robin policy to each of the m lists. As an object is seen through sorted access in some list, do random access to the other lists to find all its remaining local scores, and compute its overall score. Maintain a set Y containing the k objects whose overall scores are the highest among all the objects seen so far.
2. For each list $L_i$, let $s_i$ be the bottom score of $L_i$, which is the last local score seen under sorted access in $L_i$. Define the threshold value $\tau$ to be $\tau = g(s_1, s_2, \cdots, s_m)$.
3. Halt when $\lambda \geq \tau$, where $\lambda = \min\{g(t) \mid t \in Y\}$.
```

Figure 2. Threshold Algorithm

Let us show how TA works with the following example.

Example 1 Assume *n*=5, *m*=3, *k*=1, *g*=sum. The dataset is illustrated in Fig. 1. First, TA finds T_2 under sorted access over list L_1, then TA retrieves its second and third local scores through random access over L_2 and L_3, and then computes its overall score as $g(T_2)$ = 2. Since τ =2.9 (s_i is initialized to be 1), TA continues sorted accessing L_2. In the second sorted access, TA finds T_1, then TA retrieves its first and third local scores through random access over L_1 and L_3, and computes its overall score as $g(T_1)$ = 1.4. The current result set Y still contains T_2, but τ is updated to be 2.7. Since $\tau > g(T_2)$, TA goes on performing sorted access on L_3. This process is repeated until TA performs two sorted accesses on L_1 and one sorted access on both L_2 and L_3, where $\tau = 2.0 \leq \lambda = g(T_2)$. So, in this example, TA finds top-1 result with 4 sorted accesses and 8 random accesses.

III. A Naive Parallel Threshold Algorithm

In this section, we introduce a naive parallel-TA (PTA) algorithm for top-*k* queries processing, and analyze its speedup ratio. Specifically, we show PTA algorithm in section III-A. Thereafter, in section III-B, we analyze the speedup ratio of PTA.

A. PTA Algorithm

PTA algorithm is described in Fig. 3. Assume we have *P* processors, denoted as $p_0, p_1, \cdots, p_{P-1}$. PTA evenly partitions the original dataset *D* into *P* subdatasets, denoted as $D_0, D_1, \cdots, D_{P-1}$. Processor p_i finds top-*k* results among sub-dataset D_i using TA algorithm. Then we merge the results returned by all the processors to obtain the final top-*k* answers. The partition is completed offline.

The following theorem guarantees that PTA finds top-*k* answers for top-*k* queries with any positive integer *k* and increasingly monotonic aggregate function.

Theorem 1 *If the aggregate function is increasingly monotonic, then PTA finds exact top-k answers among the original dataset D.*

Proof: For $0 \le i \le P-1$, Let Y_i contain the top-*k* results of subdataset D_i and Y contain the final top-*k* results. Let s_{ij} denote the jth bottom value of subdataset D_i, $0 \le i \le P-1$ and $1 \le j \le m$. Let $\lambda_i = \min\{g(t) \mid t \in Y_i\}$ and τ_i denote the threshold value of subdataset D_i when PTA halts, where $0 \le i \le P-1$. And let $\lambda = \min\{g(t) \mid t \in Y\}$. According to the stopping rule of TA, we have $\tau_i \le \lambda_i$ when processor p_i halts. We have to prove that for any object $t \in D - Y$, it satisfies that $g(t) \le \lambda$. There are two cases depending on whether t is seen or not when PTA halts.

Case 1: t is seen when PTA halts. In this case, we can easily get $g(t) \le \lambda$ since Y contains the k objects with the highest scores having been seen when PTA halts.

Case 2: t is not seen when PTA halts. Assume $t \in D_i$, $0 \le i \le P-1$. Since t is not seen, we have $s_j(t) \le s_{ij}$ for $1 \le j \le m$. In addition, the aggregate function g is increasingly monotonic, so we have $g(t) \le g(s_{i1}, s_{i2}, \cdots, s_{im}) = \tau_i$. When PTA halts, we have $\tau_i \le \lambda_i$. And $\lambda_i \le \lambda$. Hence, we get $g(t) \le \tau_i \le \lambda_i \le \lambda$, i.e. $g(t) \le \lambda$, as desired. □

Parallel Threshold Algorithm (PTA)

1. Evenly partition the original dataset *D* into *P* sub-datasets, denoted as $D_0, D_1, \cdots, D_{P-1}$. This step is completed offline.
2. Processor p_i uses TA to find top-*k* answers in D_i, and stores them in Y_i, for $0 \le i \le P-1$.
3. Find the final top-*k* results among Y_i, and output them.

Figure 3. Parallel Threshold Algorithm

B. Analysis of PTA

Let a_{si} denote the number of sorted accesses performed by processor p_i, then the middleware cost of p_i is $cost(p_i) = a_{si}(c_s + (m-1)c_r)$. The cost of PTA is $cost(\text{PTA}) = \max\{cost(p_0), cost(p_1), \cdots, cost(p_{P-1})\} = \max\{a_{s0}, a_{s1}, \cdots, a_{s(P-1)}\} \times (c_s + (m-1)c_r)$. Let a_s be the number of sorted accesses needed by TA to find top-*k* results among the original dataset *D*, then the cost of TA is $cost(\text{TA}) = a_s(c_s + (m-1)c_r)$. Since the speedup ratio of PTA is $cost(\text{TA}) / cost(\text{PTA})$, we have

$$\text{speedup ratio of PTA} = \frac{a_s}{\max\{a_{s0}, a_{s1}, \cdots, a_{s(P-1)}\}} \qquad (1)$$

Equation (1) tells us that the speedup ratio of PTA is determined by the largest number of sorted accesses performed by the *P* processors and the number of sorted access performed by TA. The speedup ratio of EPTA can also be calculated using (1). In our experiments, we will use (1) to measure the speedup ratios of PTA and EPTA.

Clearly, the middleware cost of PTA is $O(nm^2 / P)$ since the middleware cost of each processor is $O(nm^2 / P)$.

IV. Enhanced Parallel Threshold Algorithm

First, we introduce some definitions in section IV-A. Thereafter, in section IV-B, we describe our enhanced parallel threshold algorithm (EPTA). Finally, in section IV-C, we analyze the cost of EPTA.

A. Definitions

In the following, we first introduce the definition of dominate and propose an observation about it. Then, we define the degree of domination and discuss its relation to top-*k* queries.

Definition 1 **Dominate** [11] *We say object t_1 dominates t_2 or t_2 is dominated by t_1 if and only if they satisfy two conditions: (1) for each $i \in \{1, 2, \cdots, m\}$, $s_i(t_1) \ge s_i(t_2)$. (2) there exists at least one number $j \in \{1, 2, \cdots, m\}$ satisfying $s_j(t_1) > s_j(t_2)$*

Our definition of dominate is different from that in [11], since we use $\ge$ (or >) instead of $\le$ (or <). However, there does not exist essential differences between them.

Observation 1: *If object t_1 dominates object t_2 and the aggregate function is increasingly monotonic, then we have $S_{t_1} \ge S_{t_2}$.*

Proof: We can easily get the correctness of the observation according to the definition of dominate. □

Definition 2 **degree of domination** [12]. *If some object t is dominated by i other objects, we say the degree of domination of t is i, denoted as $dd(t) = i$.*

This definition provides us a kind of method to classify objects, i.e. we can classify objects into *n* categories by their degree of domination. Based on the classification of the objects, we can partition the original dataset D into n subdatasets, denoted as $D_0, D_1, \cdots, D_{n-1}$ (some subdatasets may be empty), where $\forall t \in D_i$, satisfying $dd(t) = i$. The partition is pre-computed. And we can use BFA [12] or Bitmap [17] to complete the pre-computation. The following observation reduces our accessing scope to $D_0, D_1, \cdots, D_{k-1}$ when answering top-*k* queries with increasing monotonic aggregate functions.

Observation 2: *If the aggregate function is increasingly monotonic, then the top-k objects of subdatasets* $D_0, D_1, \cdots, D_{k-1}$ *are the top-k objects of the original dataset D.*

Proof: For any object t, $t \notin D_0 \cup D_1 \cup \cdots \cup D_{k-1}$, there exists at least k other objects that belong to $D_0 \cup D_1 \cup \cdots \cup D_{k-1}$ and dominate t. According to observation 1, we know that the overall scores of these objects are no less than that of t. So the top-*k* objects of subdatasets $D_0, D_1, \cdots, D_{k-1}$ are the top-*k* objects of the original dataset D for any increasingly monotonic aggregate function. □

This observation enlightens us that we only need to distribute $D_0, D_1, \cdots, D_{k-1}$ onto P processors in order to find top-*k* results for any increasingly monotonic aggregate function, which is the main idea of the following EPTA algorithm.

B. EPTA Algorithm

Based on the observations in the above section, we propose EPTA (Enhanced-PTA) algorithm in this section. Assume we have P processors denoted as $p_0, p_1, \cdots, p_{P-1}$. EPTA algorithm works in three phases, i.e. distributing phase, computing phase and merging phase. In distributing phase, EPTA distributes the subdatasets $D_0, D_1, \cdots, D_{k-1}$ onto the P processors using a heuristic distribution algorithm. In computing phase, for $0 \le i \le P-1$, processor p_i runs an algorithm designed by us to compute top-($k - d_i$) results among the subdatasets distributed onto it, where d_i is the minimum degree of domination of the objects distributed onto p_i. In merging phase, EPTA obtains the final top-*k* answers by merging the results returned in the second phase.

Algorithm 1: Used in Distributing Phase

```
1:  Let sum be the sum of the cardinality of $D_0, D_1, \cdots, D_{k-1}$.
2:  sum= sum/P
3:  j=k-1
4:  for i=P-1 down to 0 do
5:     sum_1=0
6:     while sum_1<sum and j>i-1 do
7:        Distribute $D_j$ onto $p_i$
8:        sum_1= sum_1+$|D_j|$
9:        j=j-1
10:    end while
11: end for
```

Figure 4. Algorithm 1: used in distributing phase of EPTA

Distributing Phase In this phase, EPTA distributes subdatasets $D_0, D_1, \cdots, D_{k-1}$ onto P (we assume $P<k$) processors using algorithm 1 shown in Fig. 4. Algorithm 1 tries to evenly distribute subdatasets $D_0, D_1, \cdots, D_{k-1}$ onto the P processors.

Computing Phase In this phase, each processor executes algorithm 2 described in Fig. 5. Assume the input is subdatasets $D_{i_0}, D_{i_1}, \cdots, D_{i_\sigma}$, which are arrayed in increasing order respect to the degree of domination of the objects in them. Then algorithm 2 finds top-*k* answers among $D_{i_0}, D_{i_1}, \cdots, D_{i_\sigma}$ for any positive integer k and any increasingly monotonic aggregate function.

Algorithm 2: Used in Computing Phase

```
1. Run TA on $D_{i_0}$ to find top-k objects in $D_{i_0}$, and store the
   results in $Y$, which is a set containing current top-k
   answers. Let $\lambda$ =min{ $g(t) \mid t \in Y$ }.
2. $j := 1$
3. Do sorted access in a round-robin policy to each of the
   m lists of $D_{i_j}$. As an object is seen through sorted access
   in some list, do random access to the other lists to find
   all its remaining local scores, and compute its overall
   score. Then update $s_{1j}, s_{2j}, ..., s_{mj}$ (bottom scores of $D_{i_j}$),
   $\tau_j$ (the threshold value of $D_{i_j}$), $Y$ and $\lambda$. When $\lambda \ge \tau_j$, go
   to step 4.
4. $j := j+1$. If $j \le \sigma$, then go to step 3, or go to step 5
5. Output $Y$.
```

Figure 5. Algorithm 2: used by each processor in the computing phase

The following theorem shows us the correctness of algorithm 2.

Theorem 2 *If the aggregate function* g *is increasingly monotonic, then algorithm 2 finds the exact top-k queries results among subdatasets* $D_{i_0}, D_{i_1}, \cdots, D_{i_\sigma}$ *for any positive integer k.*

Proof: Let Y contain the top-*k* results. For any object t outside Y, we need to prove that $g(t) \le \lambda$, where λ = min { $g(t) \mid t \in Y$ }. There are two cases depending on whether t is seen or not when algorithm 2 halts.

Case 1: t is seen when algorithm 2 halts. In this case, we can easily get $g(t) \le \lambda$ since Y contains the k objects with the highest scores having been seen when it halts.

Case 2: t is not seen when algorithm 2 halts. Assume $t \in D_{i_j}$, $1 \le j \le \sigma$. Since t is not seen, we have $s_i(t) \le s_{ij}$. In addition, the aggregate function g is increasingly monotonic, so $g(t) \le g(s_{1j}, s_{2j}, \cdots, s_{mj}) = \tau_j$. And $\lambda \ge \tau_j$ when algorithm 2 halts. Hence, we have $g(t) \le \lambda$, as desired. □

In the computing phase, processor p_i executes algorithm 2 to find the top-($k - d_i$) results among the subdatasets distributed on it, where d_i is the minimum degree of domination of the objects distributed onto p_i. This is because, according to observation 2, top-d_i results are among the objects whose degree of domination are less than d_i .

Merging Phase In this phase, EPTA merges the results returned in the second phase to obtain the final top-*k* answers.

Now, we can describe EPTA algorithm. EPTA algorithm is shown in Fig. 6. With observation 2 and theorem 2, we can easily prove the correctness of EPTA. The speedup ratio of EPTA can be calculated using (1).

```
Enhanced Parallel Threshold Algorithm (EPTA)
1. Run algorithm 1 to distribute subdatasets $D_0, D_1, \cdots, D_{k-1}$ onto the $P$ processors.
2. Processor $p_i$ runs algorithm 2 to compute top-($k-d_i$) results among the subdatasets distributed onto it, where $d_i$ is the minimum degree of domination of the objects distributed onto $p_i$.
3. Obtain the final top-$k$ results by merging the results returned in step 2.
```

Figure 6. Enhanced parallel threshold algorithm

C. Cost Analysis

Our analysis is under PRAM-CRCE model. The following theorem tells us the upper bound of the average middleware cost of EPTA under any distribution of the scores in the lists.

Theorem 3 *For any top-k queries, under any distribution of the scores in the lists, the average middleware cost of EPTA algorithm is:*

$$E(cost(\text{EPTA})) = O(m^2 \sum_{i=0}^{k-1} E(|D_i|)) \qquad (2)$$

Proof: For $0 \le i \le P-1$, let random variable n_i denote the number of objects distributed onto processor p_i . Let variable $cost_i$ denote the middleware cost of processor p_i . And let $cost(\text{EPTA})$ denote the cost of algorithm EPTA. Clearly, we have $cost_i \le n_i m(c_s + (m-1)c_r) = cn_i$, where $c = m(c_s + (m-1)c_r) = O(m^2)$. And, under PRAM-CRCW model, we have $cost(\text{EPTA})$ = $\max\{cost_0, cost_1, \cdots, cost_{P-1}\} \le \max\{cn_0, cn_1, \cdots, cn_{P-1}\} \le c\sum_{i=0}^{P-1} n_i$. So, we have $E(cost(\text{EPTA})) \le c\sum_{i=0}^{P-1} E(n_i)$. Since in EPTA, we distribute subdatasets $D_0, D_1, \cdots, D_{k-1}$ onto the P processors, we have $\sum_{i=0}^{P-1} E(n_i) = \sum_{i=0}^{k-1} E(|D_i|)$. So, we obtain $E(cost(\text{EPTA})) \le c\sum_{i=0}^{k-1} E(|D_i|) = O(m^2 \sum_{i=0}^{k-1} E(|D_i|))$, as desired. □

Equation (2) is a loose upper bound of the average middleware cost of EPTA. It's one of our future works to establish a tighter upper bound.

From (2), we know that we have to calculate $E(|D_i|)$ in order to estimate the average middleware cost of EPTA. Some previous papers have tried to estimate the average cardinality of the *skyline* (it's the same as subdataset D_0) of the dataset. Under the assumption that scores in different lists are independently distributed, Bentley et al. [2] proves that $E(D_0) = O((\ln n)^{m-1})$. However, it's a loose upper bound. Buchta [18] establishes that $E(D_0) = \Theta((\ln n)^{m-1} / (m-1)!)$. But their proofs are restricted to the cases where no two objects share the same local score in any list. Chaudhuri et al. [1] proves that this restriction does not influence the expected cardinality of subdataset D_0 . However, the previous works only estimate the average cardinality of D_0 . We need to estimate the average cardinality of every subdataset, which is more challenging. In the following, we get $O(E(|D_i|)) = O((\ln n)^{m-1} / (m-1)!)$ by deducing $E(|D_i|) \le E(|D_0|)$ $(1 \le i \le n-1)$ under the assumption that the scores in different lists are independent, based on which, we obtain $E(cost(\text{EPTA})) = O(km^2 (\ln n)^{m-1} / (m-1)!)$.

Let $A_i(n,m) = E(D_i)$. We assume that the n objects are randomly and independently chosen from the same distribution. Let the set of m random variables $X_1, X_2, \cdots, X_m$ continuously ranging in the interval [0, 1] represent the local scores in the m lists. Let $F(x_1, x_2, \cdots, x_m)$ denote the joint distribution function, and $f(x_1, x_2, \cdots, x_m)$ denote the joint density function of the m variables. In vector notation, we rewrite them as $F(\overline{X})$ and $f(\overline{X})$, where $\overline{X} = (x_1, x_2, \cdots, x_m)$. Then we have the following theorem, which formulates $A_i(n,m)$.

Theorem 4 *Let* $A_i(n,m)$ *denote the average cardinality of* D_i *, then we have*

$$A_i(n,m) = n\binom{n-1}{i}\int_{[0,1]^m} f(\overline{X})P^i(\overline{X})(1-P(\overline{X}))^{n-i-1} d\overline{X} \qquad (3)$$

Where $P(\overline{X}) = \int_{\overline{Y} \ge \overline{X}} f(\overline{Y}) d\overline{Y}$.

Proof: For any object t whose local scores are vector $\overline{X} = (x_1, x_2, \cdots, x_m)$, the probability that another object t' dominates it is $P(\overline{X})$ and the probability that t' does not dominate it is $(1-P(\overline{X}))$. In order to make the degree of domination of t be i , i out of the other $(n-1)$ objects should dominate t and the rest shouldn't dominate t . So, the probability $\Pr\{dd(t) = i\} = \binom{n-1}{i} P^i(\overline{X})(1-P(\overline{X}))^{n-i-1}$. Since object t itself comes from a distribution with density function $f(\overline{X})$, the probability that a randomly chosen object belongs to D_i is $\int_{[0,1]^m} f(\overline{X})\binom{n-1}{i} P^i(\overline{X})(1-P(\overline{X}))^{n-i-1} d\overline{X}$. So the average cardinality of D_i , i.e. $A_i(n,m)$, is formulated as (3), as desired. □

Equation (3) is established under any distribution of the scores in the lists. The following theorem formulates $A_i(n,m)$

under the assumption that scores in different lists, i.e. $X_1, X_2, \cdots, X_m$, are independent.

Theorems 5 *If the scores in different lists are independent, then we have*

$$A_i(n,m) = n\binom{n-1}{i}\int_{[0,1]^m} (1-x_1x_2\cdots x_m)^{n-i-1} x_1^i x_2^i \cdots x_m^i \, dx_1\cdots dx_m \quad (4)$$

Proof: Since $X_1, X_2, \cdots, X_m$ are independent, $f(\overline{X})$ can be written as $f_1(x_1)f_2(x_2)\cdots f_m(x_m)$ and $F(\overline{X})$ can be written as $F_1(x_1)F_2(x_2)\cdots F_m(x_m)$, where $f_j(x_j)$ is the density function and $F_j(x_j)$ is the distribution function of X_j, where $1 \le j \le m$. $P(\overline{X})$ $= \int_{\overline{Y}\ge\overline{X}} f(\overline{Y})d\overline{Y} = \prod_{j=1}^{m}(1-F_j(x_j))$. Moreover, $f_j(x_j) = -(1-F_j(x_j))'$ since X_j is continues. Substituting these to (3), and using a change of variable, we have the correctness of theorem 5. □

When $i=0$, $A_0(n,m) = n\int_{[0,1]^m}(1-x_1x_2\cdots x_m)^{n-1}dx_1dx_2\cdots dx_m$, which is the same as that in [1].

After expanding $(1-x_1x_2\cdots x_m)^{n-i-1}$, we can calculate (4) as follows:

$$A_i(n,m) = n\binom{n-1}{i}\int_{[0,1]^m}(1-x_1x_2\cdots x_m)^{n-i-1}x_1^ix_2^i\cdots x_m^i dx_1\cdots dx_m$$

$$= n\binom{n-1}{i}\int_{[0,1]^m}\sum_{j=0}^{n-i-1}\binom{n-i-1}{j}(-1)^j(x_1x_2\cdots x_m)^{j+i}dx_1dx_2\cdots dx_m$$

$$= n\binom{n-1}{i}\sum_{j=0}^{n-i-1}\binom{n-i-1}{j}(-1)^j\left(\int_{[0,1]}x^{i+j}dx\right)^m$$

$$= n\binom{n-1}{i}\sum_{j=0}^{n-i-1}\binom{n-i-1}{j}(-1)^j\frac{1}{(i+j+1)^m}$$

$$= \sum_{j=1}^{n-i}\binom{i+j-1}{i}\binom{n}{i+j}(-1)^{j+1}\frac{1}{(i+j)^{m-1}}$$

So, we have

$$A_i(n,m) = \sum_{j=1}^{n-i}\binom{i+j-1}{i}\binom{n}{i+j}(-1)^{j+1}\frac{1}{(i+j)^{m-1}} \quad (5)$$

For $n \ge 1, m \ge 1, 0 \le i \le n-1$

With (5), we can easily get $A_i(i+1,m) = \frac{1}{(i+1)^{m-1}}$ and $A_j(n,1) = 1$, where $i \ge 0$ and $0 \le j \le n-1$. In order to analyze (5), we define a set of generating functions $G_i(n,z)$ as

$$G_i(n,z) = \sum_{m=0}^{\infty}A_i(n,m+1)z^m \text{, for } 0 \le i \le n-1$$

By (5), we have

$$(1-z/n)G_i(n,z) = \sum_{m=0}^{\infty}A_i(n,m+1)z^m$$

$$= (1-z/n)\sum_{m=0}^{\infty}\sum_{j=1}^{n-i}\binom{i+j-1}{i}\binom{n}{i+j}(-1)^{j+1}\frac{1}{(i+j)^m}z^m$$

$$= \sum_{m=0}^{\infty}\sum_{j=1}^{n-i}\binom{i+j-1}{i}\binom{n}{i+j}(-1)^{j+1}\frac{1}{(i+j)^m}z^m$$

$$-\sum_{m=0}^{\infty}\sum_{j=1}^{n-i}\binom{i+j-1}{i}\binom{n-1}{i+j-1}(-1)^{j+1}\frac{1}{(i+j)^{m+1}}z^{m+1}$$

$$=1+\sum_{m=1}^{\infty}\sum_{j=1}^{n-i}\binom{i+j-1}{i}\left(\binom{n}{i+j}-\binom{n-1}{i+j-1}\right)(-1)^{j+1}\frac{1}{(i+j)^m}z^m$$

$$= G_i(n-1,z)$$

So we get $G_i(n,z) = G_i(n-1,z)/(1-z/n)$. Since $A_i(i+1,m)$ $= \frac{1}{(i+1)^{m-1}}$ and $A_i(j,m) = 0$ for $i \ge 0$ and $j \le i$, we have $G_i(i+1,z) = 1/(1-z/(i+1))$ and $G_i(j,z) = 0$ for $i \ge 0$ and $j \le i$. So we obtain

$$G_i(n,z) = \prod_{j=i+1}^{n}\frac{1}{1-z/j} \quad (6)$$

Equation (6) is (36) of Knuth [3, Sec. 1.2.9] with $x_j = 0$ for $1 \le j \le i$ and $x_j = 1/j$ for $i+1 \le j \le n$. Define

$$H_i^{(r)}(n) = \sum_{j=i+1}^{n}1/j^r \text{, for } 0 \le i \le n-1$$

By Knuth's analysis, the coefficient of z^m in $G_i(n,z)$ is

$$\sum_{\substack{k_1,k_2,\cdots,k_m\ge 0\\ k_1+2k_2+\cdots+mk_m=m}}\frac{H_i^{(1)}(n)^{k_1}}{1^{k_1}k_1!}\frac{H_i^{(2)}(n)^{k_2}}{2^{k_2}k_2!}\cdots\frac{H_i^{(m)}(n)^{k_m}}{m^{k_m}k_m!} \quad (7)$$

According to definition, (7) is $A_i(n,m+1)$. Since $H_i^{(r)}(n) \le H_{i-1}^{(r)}(n)$ for $1 \le i \le n-1, r \ge 1$, we have

$$A_i(n,m) \le A_{i-1}(n,m) \text{, for } 1 \le i \le n-1 \quad (8)$$

Equation (8) shows $A_i(n,m)$ is monotonic respect to i. Buchta [18] established that $A_0(n,m) = \Theta((\ln n)^{m-1}/(m-1)!)$. So, we have

$$A_i(n,m) = O((\ln n)^{m-1}/(m-1)!) \text{, for } 0 \le i \le n-1 \quad (9)$$

So, we have the following theorem, which is an upper bound of the average middleware cost of EPTA algorithm.

Theorem 6 *If the scores in different lists are independently distributed, then the average middleware cost of EPTA is*

$$E(cost(\text{EPTA})) = O(km^2(\ln n)^{m-1} / (m-1)!) \qquad (10)$$

Proof: We can easily get the correctness by combining (2) and (9).□

This upper bound is a loose one. From the above analysis, we can know that the cardinality of subdataset D_i is a binomial distribution under the assumption that the scores in different lists are independently distributed. It's possible to get a tighter upper bound of the average middleware cost of algorithm EPTA. Actually, it's one of our future works.

V. EXPERIMENTS

In this section, we do extensive experiments to evaluate the performance of our algorithms, i.e. PTA and EPTA. Our algorithms are implemented in C++ and OpenMP. We perform our experiments on a 2-CPU server with shared memory 8GB; each CPU is 4-core Intel Xeon E5430 2.66GHz. We use both synthetic datasets and real dataset in our experiments. The metrics we measure is the speedup ratio. According to (1), we only need to count the number of sorted access performed by TA, PTA and EPTA in order to calculate the speedup ratio. First, in section V-A, we show the experimental results on synthetic datasets. Then in section V-B, we show the experimental results over real datasets. Finally, we summarize our experiments in section V-C.

A. *Experiments with Synthetic Datasets*

Datasets Generation We do experiments on three synthetic datasets. All generated local scores belong to the interval [0, 1]. The three synthetic datasets are produced to model different input scenarios; they are UI, NI and CO, respectively. UI contains datasets in which objects' local scores are uniformly and independently generated for the different lists. NI contains datasets in which objects' local scores are normally and independently generated for the different lists. CO, generated using the same methodology in [11], contains datasets in which objects' local scores are correlated. In other words, the ith local score $s_i(t)$ of an object t is very close to $s_j(t)$ with high probability, where $i \neq j$. To generate an object t, first, a number u_t from 0 to 1 is selected using a Gaussian distribution centered at 0.5 with variance 1. t's local scores are then generated by a Gaussian distribution centered at u_t with variance 0.01.

For synthetic datasets, our default settings for different parameters are shown in Table 1. In our tests, the default number of data items in each list is 1,000,000, i.e. n=1,000,000. Typically, users are interested in a small number of top answers, thus we set k = 50 as k's default value. Like many previous works on top-k query processing, such as [12][19], we choose the aggregate function as the sum of the local scores. In one third of our tests, the number of lists, i.e. m, is a varying parameter. When m is a constant, we set it to 3 since most previous works evaluate their algorithms on datasets with 3 lists.

TABLE I. DEFAULT VALUES OF EXPERIMENTAL PARAMETERS

Parameters	Default Values
The number of objects, i.e. n	1,000,000
The number of lists, i.e. m	3
The number of results returned, i.e. k	50
The number of processors, i.e. P	8
Aggregate function	summation

Experimental Results Fig. 7 shows the experimental results when k, the number of results returned, varies from 10 to 100 with step 10 and other parameters set as those in Table 1. From the results, we know that the speedup ratios of EPTA are orders of magnitude higher than those of PTA over all the datasets we consider. Over the three datasets considered, the speedup ratios of PTA almost keep unchanged as k increases. However, the speedup ratios of EPTA decrease as k increases. The reason is that the number of objects distributed onto each processor in PTA has nothing to do with k while EPTA has to distribute more subdatasets onto the processors as k increases.

Fig. 8 illustrates the experimental results in which the parameters are set as those in Table 1 except that m, the number of lists, varies from 2 to 8 with step 1. The results show us that the speedup ratios of EPTA are significantly higher than those of PTA over the considered datasets. The speedup ratios of PTA increase slightly as m increases while the speedup ratios of EPTA increase, and then decrease as m increases. Interestingly enough, the speedup ratios of EPTA are maximal when m=3 over all datasets we consider. It's one of our future works to explore the theoretical reason why this happens. The reason why the speedup ratios of EPTA decrease as m increases is that the k subdatasets distributed onto the P processors contain more and more objects as m increases.

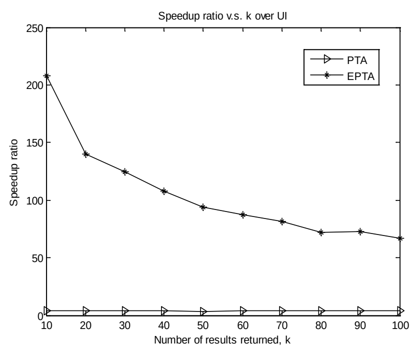
Fig. 9 illustrates the experimental results when the parameters are set as those in Table 1 except that the dataset size, i.e. n, varies. The results tell us that the speedup ratios of EPTA are orders of magnitude higher than those of PTA. The speedup ratios of PTA almost keep unchanged while the speedup ratios of EPTA increase fast as n increases.

From the analysis above, we know that the speedup ratios of EPTA are orders of magnitude higher than those of PTA over all the datasets considered. The reason is that EPTA only distributes k subdatasets onto the P processors while PTA distributes the whole original dataset onto the processors.

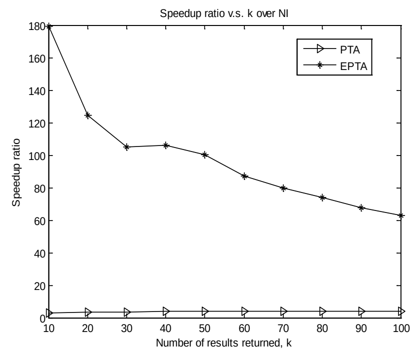
B. *Experiments with Real Datasets*

Description of Datasets For real datasets, we choose El Nino dataset[1] and Forest Cover (FC) dataset[2]. El Nino dataset contains 93935 objects with 3 lists, and FC dataset contains 581012 objects with 4 lists. El Nino contains oceanographic and surface meteorological readings taken from a series of

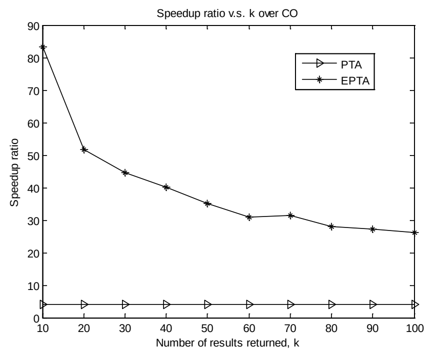
[1]From UCI KDD. http://kdd.ics.uci.edu/databases/el_nino/el_nino.html
[2]From UCI KDD. http://kdd.ics.uci.edu/databases/covertype/covertype.html



(a) over UI

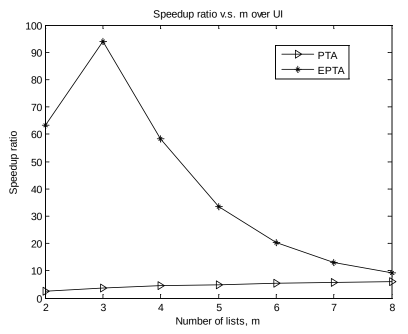


(b) over NI

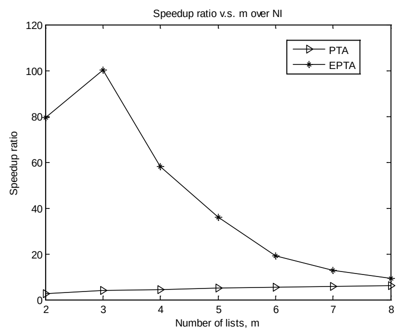


(c) over CO

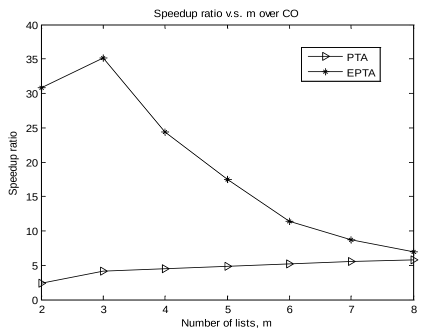
Figure 7. Speedup ratios v.s. *k* over synthetic datasets, *n*=1,000,000, *m*=3, *P*=8



(a) over UI

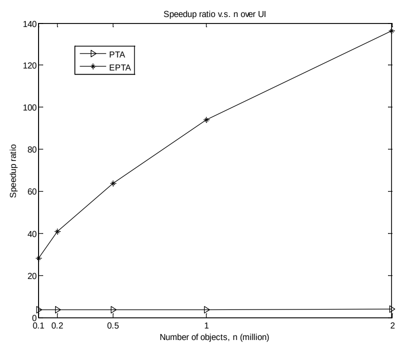


(b) over NI

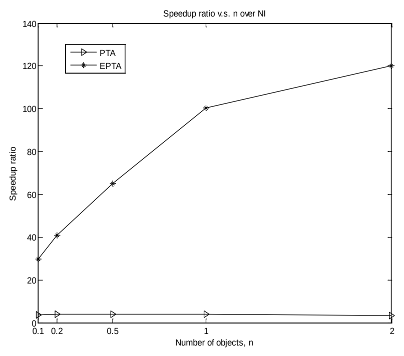


(c) over CO

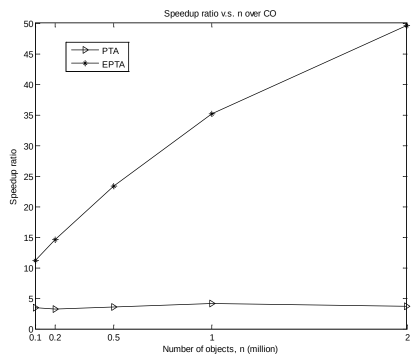
Figure 8. Speedup ratios v.s. *m* over synthetic datasets, *n*=1,000,000, *k*=50, *P*=8



(a) over UI

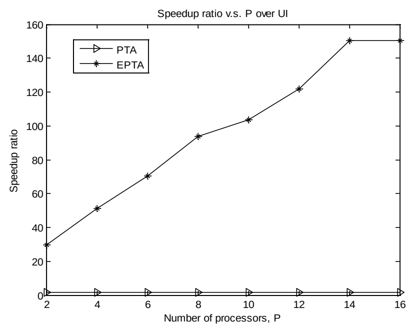


(b) over NI

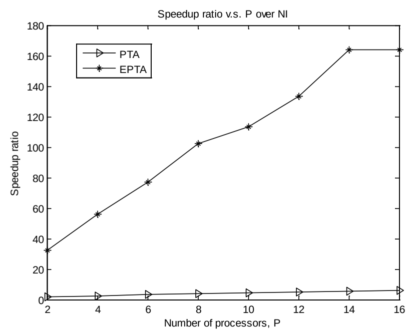


(c) over CO

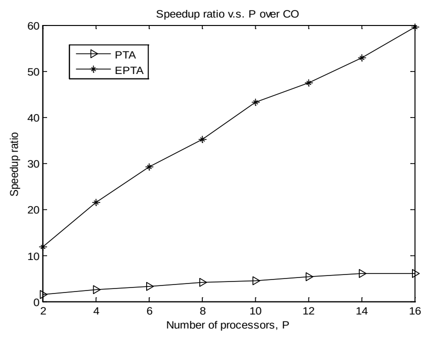
Figure 9. Speedup ratios v.s. *n* over synthetic datasets, *m*=3, *k*=50, *P*=8



(a) over UI



(b) over NI

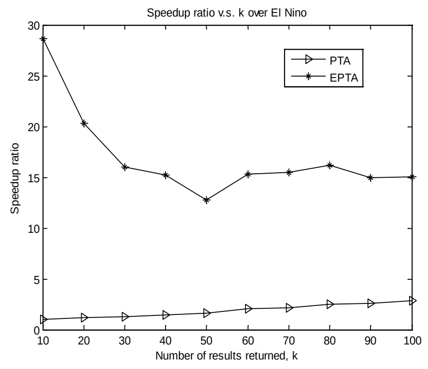


(c) over CO

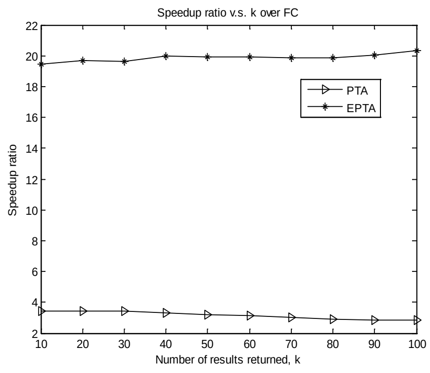
Figure 10. Speedup ratios v.s. *n* over synthetic datasets, *m*=3, *k*=50, *P*=8

buoys positioned throughout the equatorial Pacific. The data is expected to aid in the understanding and prediction of El Nino/Southern Oscillation (ENSO) cycles. We neglect the objects missing some fields. The remaining dataset contains 93935 objects. We chose 3 lists to test our algorithms. FC contains 581012 forest land cells (i.e. objects), having four attributes (i.e. lists): horizontal distance to nearest surface water features, vertical distance to nearest surface water features, horizontal distance to nearest roadways, and horizontal distance to nearest wildfire ignition points. For both real datasets, we normalize the dataset with the formula: $\frac{s_i(t) - Min}{Max - Min}$, where $s_i(t)$ is t 's ith local score. In our experiments, the number of results returned, i.e. *k*, varies from 10 to 100 with step 10.

Experimental Results Fig. 11 shows the experimental results over El Nino dataset (Fig. 11 (a)) and FC dataset (Fig. 11 (b)). From the results, we know that the speedup ratio of EPTA is significantly higher than that of PTA. The reason is that EPTA only distributes objects whose degree of domination

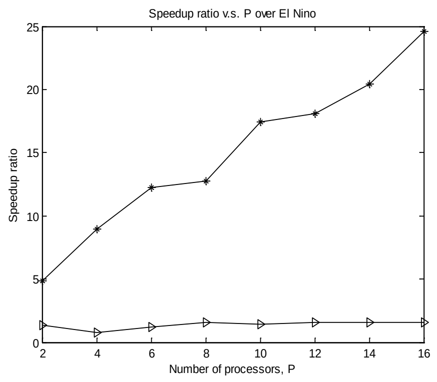


(a) over El Nino

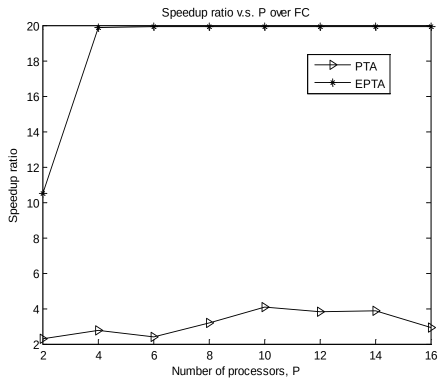


(b) over FC

Figure 11. Speedup ratios v.s. *k* over real datasets, *P*=8



(a) over El Nino



(b) over FC

Figure 12. Speedup ratios v.s. *P* over real datasets, *k*=50

is less than *k* onto processors while PTA distributes the whole dataset onto processors. For El Nino dataset, the speedup ratio of EPTA decreases as the number of results returned (i.e. *k*) increases while the speedup ratio of PTA increases slightly as *k* increases. However, for FC dataset, the speedup ratio of EPTA almost keeps unchanged as *k* increases while the speedup ratio of PTA decreases slightly as *k* increases.

C. Summary

Our experiments illustrate that the speedup ratios of EPTA are orders of magnitude higher than those of PTA over the synthetic and real datasets considered. This can be explained by the fact that PTA distributes the whole original dataset onto the *P* processors while EPTA first partitions the original dataset into *n* subdatasets according to the degree of domination of the objects, and only distributes *k* subdatasets onto the processors for any top-*k* query with monotonically increasing aggregate function.

VI. RELATED WORKS

Several papers are related to our analysis in section IV-C. In computational geometry community, Bentley et al.[2] analyzed the average number of *maxima* (i.e. subdataset D_0) of a dataset. Under the assumption that the local scores in different lists are independently distributed, they got $E(|D_0|) = O((\ln n)^{m-1})$. Buchta [18] got a tighter bound, that is, $E(D_0) = \Theta((\ln n)^{m-1} / (m-1)!)$. But their proofs are restricted to the cases where no two objects share the same local score in any list. When the term *maxima* was introduced into database area, it was renamed as *skyline* of the dataset by database community. Chaudhuri et al. [1] proves that Bently's and Buchta's estimations are also right in the cases some objects share the same local scores in some lists.

The previous works only estimate the average cardinality of subdataset D_0 (named as *maxima* in computational geometry community and *skyline* in database community). We need to estimate the average cardinality of every subdataset, which is much more challenging. We prove that $E(|D_i|) \le E(|D_0|)$ ($1 \le i \le n-1$) under the assumption that the local scores in different lists are independently distributed. So, we have $E(D_i) = O((\ln n)^{m-1}/(m-1)!)$, based on which, we obtain the average middleware cost of EPTA is $O(km^2(\ln n)^{m-1}/(m-1)!)$.

VII. Conclusions and Future Works

In this paper, we propose parallel algorithms, i.e. PTA and EPTA, to process top-*k* queries with any positive integer *k* and any monotonically increasing aggregate function. PTA evenly partitions the original dataset into *P* subdatasets. Each processor finds top-*k* results of one corresponding subdataset using TA algorithm. Then these results are merged to get the final top-*k* answers. The middleware cost of PTA is $O(nm^2/P)$ under PRAM-CRCW model. EPTA is based on the partition of the dataset according to the objects' degree of domination. And it works in three phase. In the first phase, it distributes *k* subdatasets on to *P* processors using a heuristic algorithm. In the second phase, processor p_i finds the top-($k-d_i$) results among the subdatasets distributed onto it, where d_i is the minimum degree of domination of the objects distributed onto p_i. In the third phase, EPTA obtains the final top-*k* results by merging the results returned in the second phase. The average middleware cost of EPTA is $O(km^2(\ln n)^{m-1}/(m-1)!)$ under the assumption that scores in different lists are independent.

In both PTA and EPTA, processors do not communicate until they find their own query answers among the corresponding subdatasets. Actually, processors can communicate with each other and merge the local results before they find the whole local result set so as to halt earlier. In this scenario, the number of accesses performed by each processor may decrease. However, the communication cost will correspondingly increase. So, in the future, we plan to manage the tradeoff between them.

References

[1] S. Chaudhuri, N. Dalvi and R. Kaushik. Robust Cardinality and Cost Estimation for Skyline Operator. *ICDE*, 2006.

[2] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *J. ACM*, 25(4), pp.536–543, 1978.

[3] D. E. Knuth. The Art of Computer Programming Volume 1: Fundamental Algorithms (Third Edition), Addison-Wesley, 1998.

[4] R.Akbarinia, E.Pacitti and P.Valduriez. Reducing network traffic in unstructured P2P systems using Top-k queries. *Distributed and Parallel Databases 19(2)*, 2006.

[5] R. Akbarinia, E. Pacitti, and P. Valduriez. Processing top-k queries in distributed hash tables. *Euro-Par Conf.*, 2007.

[6] B.Babcock and C.Olston. Distributed top-k monitoring. *SIGMOD*, 2003.

[7] P.Cao and Z.Wang. Efficient top-k query calculation in distributed networks. *PODC*, 2004.

[8] W.-T. Balke, W. Nejdl, W. Siberski and U. Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. *ICDE*, 2005.

[9] B.Kimelfeld and Y.Sagiv. Finding and approximating top-k answers in keyword proximity search. *PODS*, 2006.

[10] R. Fagin. Combining fuzzy information from multiple systems. J. Comput. System Sci., 58 (1), 1999..

[11] S.B¨orzs¨onyi, D.Kossmann and K.Stocker, "The skyline operator." *ICDE*, 2001.

[12] Z. Gong, G. Z. Sun, J. Yuan and Y. Zhong. Efficient Top-*k* Query Algorithms Using *K*-skyband Partition. *INFOSCALE*, 2009.

[13] M. Persin, J. Zobel and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. J. of the American Society for Information Science 47(10), 1996.

[14] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. ACM Transactions on Database Systems (TODS) 27(4), 2002.

[15] G.R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. ACM Transactions on Database Systems (TODS), 28(4), 2003.

[16] R. Fagin, A. Lotem and M. Naor. "Optimal aggregation algorithms for middleware". *PODS*, 2001.

[17] K. Tan, P. Eng and, B. Ooi. Efficient Progressive Skyline Computation. *VLDB*, 2001.

[18] C. Buchta. On the average number of maxima in a set of vectors. *Information Processing Letters* 33 pp.63–65, 1989.

[19] J. Yuan, G. Z. Sun, Y. Tian, G. Chen and Z. Liu. "Selective-NRA Algorithms for Top-k Queries." *APWeb/ WAIM* 2009.