Efficient Parallel Top-k Computation Algorithm using Symmetry Breaking

Chao Wu, Guang-zhong Sun, Guo-liang Chen
School of Computer Science and Technology
University of Science and Technology of China
Anhui Key Laboratory of High Performance Computing
Hefei, Anhui 230027
Email: warrior@mail.ustc.edu.cn,{gzsun,glchen}@ustc.edu.cn

***Abstract*—A key problem of relational database is to aggregate different values of the same object and find the first k objects with highest overall values. Many sequential algorithms have been proposed to solve this problem. In this paper, we propose a new parallel algorithm using symmetry breaking strategy. New algorithm is proved to be instance optimal. Experiment results on both synthetic and real data also show that new algorithm costs fewer accesses, compared to previous algorithms.**

I. Introduction

In recent years, parallel hardware has experienced a great evolution. The computing power of parallel computers has made a big step forward thanks to Multi-core technology and hyper-threading technology. Along with the hardware evolution, parallel software also get great development. Parallel computing has now expanded its landscape from original numerical circle to a large variety of aspects, such as machine learning, database software and so on [1].

Query processing is one of key functions in database software. Supposed that we are going to rent a house, the suitableness of a house can be calculated using attributes like house price, local safety and traffic convenience. The query is to combine all these attribute data together and search for the set with highest attribution scores. Since there are hundreds of megabytes or more data stored in database, it becomes a urgent request for query algorithm to process those data quickly and correctly.

The acquiring of highest attribution scores set is well known as top-k query. Top-k query has a lot of applications in information retrieval [2], data mining [3], multimedia datasets [4]. And it attracted a lot of academical enthusiasm. By different accessing patterns to database, top-k queries are divided into two kinds. One kind is query with both sorted access and random access. In this area, two famous algorithms are Fagin's algorithm (FA) and threshold algorithm(TA) [5]. The other kind is query with no random access, in consideration that random access is either forbidden or too expensive. Fagin et al. [5] proposed "No Random Access" algorithm(NRA) and proved it is instance optimal. Güntzer et al. [6] proposed an on-line algorithm "Stream-Combine"(SC), which only reports objects with exact values [7]. Nikos Mamoulis et al. [8] optimized NRA algorithm by dividing it into two phases and applying different strategies to different phases. Jing Yuan et al. [9] modified NRA algorithm into a greedy algorithm which costs less, but new algorithm is not instance-optimal. However these are all sequential algorithms, there has been little work on how to use parallel computers to process that large amount of data and accelerate top-k queries. Thus we investigated this area and proposed parallel top-k query algorithms in this paper.

The rest of paper is organized as follows. Section II gives the exact definition of top-k query. Section III proposes our PNRA and RPNRA algorithms for parallel top-k query. Section IV shows the experiment results of our algorithms on both synthetic and real data. Section V concludes the paper.

II. Problem Definition

Assume there are n objects and m attributes lists $L_1, L_2, \cdots, L_m$ in database DB. Each object x has a ID and the attributes of x are $(x_1, \cdots, x_m)$. List L_i is constituted of pairs (ID, x_i) and sorted in descending order of x_i. Using aggregation function t, the aggregation value of object x is calculated by $t(x) = t(x_1, \cdots, x_m)$. Aggregation function t is *monotone* if $t(x) \leq t(x')$ whenever $x_i \leq x'_i$ for every i. Top-k query is thus to find k objects with highest aggregation values.

There are two different types of how to access to the ith attribution of object x in database, given i [10]. The first one is *random access*, accessing the ith value of x in list L_i by using object ID in one step. The second one is *sorted access*, iteratively reading objects and their values above pair (ID, x_i) in list L_i sequentially until get the acquired value. The cost of top-k query is $rC_r + sC_s$, where r is the number of random accesses, s is the number of sorted accesses, C_r and C_s are costs of random access and sorted access respectively.

In the following paragraphs, we focus only on no random access queries. The aggregation function is given and supposed to be monotone. For any object R, the best value $B_S(R)$ is computed by assuming unseen grades to be highest possible value, the worst value $W_S(R)$ is computed by assuming unseen grades to be lowest possible value. The main idea of NRA algorithm [5] is to maintain a candidate set of objects with highest k worst value seen so far. When the minimal worst value of candidate set is no less than the maximal best value of objects which are not in candidate set, the NRA

978-0-7695-4190-7/10 $26.00 © 2010 IEEE
DOI 10.1109/ISPA.2010.47

L_1	L_2
$(R_1, 0.9)$	$(R_2, 0.5)$
$(R_2, 0.5)$	$(R_3, 0.5)$
$\cdots$	$\cdots$
$(R_{n-2}, 0.5)$	$(R_{n-1}, 0.5)$
$(R_{n-1}, 0.5)$	$(R_n, 0.5)$
$(R_n, 0.5)$	$(R_1, 0)$

Fig. 1. Example Database

algorithm terminates. The candidate set is thus the answer to top-k query by NRA algorithm.

III. PNRA Algorithm

In this section, first a query example is given to show that NRA costs more than necessary. Then we propose our PNRA algorithm and prove it is correct and instance optimal. After that we give a mathematic analysis on what conditions our PNRA algorithm is better than NRA algorithm (in access cost). Finally we propose a variant of PNRA algorithm.

A. Observation

Example Top-1 query in database shown in Figure 1.

The processing of NRA algorithm is: at depth 1, R_1 is in T_1, $M_1^{(1)} = W(R_1) = 0.9,\ B^{(1)}(R_2) = 1.4,\ B^{(1)}(R_2) > M_1^{(1)}$, NRA can not halt since stopping condition (b) is not met. At depth d from 2 to $n-1$, $R_2 \in T_1$, $M_1^{(d)} = 1,\ B^{(d)}(R_1) = 1.4,\ B^{(d)}(R_1) > M_1^{(d)}$, stopping condition (b) is not met either. Then at depth n, both stopping conditions (a) and (b) are satisfied, algorithm NRA finds the top-1 object and halts. So the cost of NRA in this query is $2n * C_s$. However, one query algorithm can find top-1 object by doing 2 sorted accesses in L_1 and n accesses in L_2. Thus the cost of this algorithm is $(n+2) * Cs$, far less than cost of NRA.

The processing of top-k query can be seen as a search for the *depth array* $\vec{D} = (d_1, d_2, \cdots, d_m)$. When any query algorithm does d_i sorted accesses in list L_i, it correctly finds the k highest-value objects. From example, it is shown that NRA uses a symmetrical strategy to generate depth array. When NRA finds top-k objects and halts at depth d, the depth it goes in each list is either d or $d-1$ which is $|d_i - d_j| = 0$ or $1 (i \neq j)$. However there are queries, each list need quite different sorted accesses to meet stopping conditions. NRA can not handle this kind of queries well.

Here are some definitions.

Definition In top-k queries, depth array $\vec{D} = (d_1, d_2, \cdots, d_m)$ is *sufficient* if stopping conditions (a) and (b) are met by performing d_i sorted accesses in L_i $(1 \leq i \leq m)$. Sufficient depth array $\vec{D} = (d_1, d_2, \cdots, d_m)$ is called *necessary* if stopping conditions (a) and (b) can not be both met by performing less than d_i sorted accesses in L_i and same sorted accesses in other lists. The *sufficient and necessary set* S is the set of all top-k sufficient and necessary depth vectors: $S = \{\vec{D} \mid \vec{D}$ is sufficient and necessary$\}$.

It is obvious that the depth array generated by NRA is sufficient.

Definition For a depth array $\vec{D} = (d_1, d_2, \cdots, d_m) \doteq (d_i)$, D_{max} is defined to be the maximal component of $\vec{D}$: $D_{max} = \max_i\{d_i\}$.

Definition A top-k query is *d-necessary* if there is at least one list needs d sorted accesses to meet the stopping conditions (a) and (b).

Lemma 1. *Assumed that algorithm NRA halts at depth d in top-k query (d objects have been accessed under sorted access in each list), then for all vectors $\vec{D}$ in top-k sufficient and necessary set S, $\min_S\{D_{max}\} = d$.*

Proof: Since NRA depth vector $\vec{D}^{(0)} = (d, d, \cdots, d)$ is sufficient, $\min_S\{D_{max}\} \leq d$. If $\min_S\{D_{max}\} < d$, then there exists a depth vector $\vec{D}^{(1)} = (d_1, d_2, \cdots, d_m)$ in S that $D_{max}^{(1)} < d$. When algorithm NRA accessed depth $D_{max}^{(1)}$, both stopping rules (a) and (b) are met Because $\vec{D}^{(1)}$ is top-k sufficient and necessary. Then NRA algorithm halts at this depth $D_{max}^{(1)} < d$, which is a contrary to lemma condition. So $\min_S\{D_{max}\} = d$. ■

Let S_d be the subset of S so that $S_d = \{\vec{D} | \vec{D} \in S, D_{max} = d\}$. S_d is not an empty set from Lemma 1. Let d_{min} be the min value among all minimum components of vectors in S_d, $d_{min} = \min_{S_d}\{D_{min}\}$.

Lemma 2. *Given aggregation function and k, if algorithm NRA halts at depth d in top-k query, then at least one list needs d sorted access to meet conditions (a) and (b).*

Proof: On the contrary, the stopping conditionss (a) and (b) are met when performing d_i sorted accesses in L_i and $d_i < d$ for all $i \in \{1, \cdots, m\}$. So depth array $\vec{D} = (d_1, d_2, \cdots, d_m)$ is top-k sufficient, which leads to $\min_S\{D_{max}\} \leq \max\{d_1, d_2, \cdots, d_m\} < d$, a contrary to Lemma 1. ■

B. Parallel-NRA Algorithm

Now we introduce our Parallel-NRA algorithm(PNRA). In order to handle those queries components of depth array differ greatly, we use symmetric breaking strategy to generate depth array. For processor P_i, our algorithm try to sorted access fewer objects in list L_i and access more objects in other lists. Assumed there are M processors($M > m$). Database DB is located in public memory pool. The sorted access cost to any list from any process is the same C_s. For processor P_i, in one super step PNRA do parallel accesses 1 depth to L_i and T depths to other lists L_j, $(j \neq i)$. At each super step d, PNRA tests the stopping conditions for every processor. Once a processor meets the conditions, it broadcasts abort command to others and return its local $T_k^{(d)}$ as results. Details of Parallel-NRA algorithm are given in Figure 2.

Theorem 1. *If aggregation function is monotone, then PNRA algorithm correctly finds the top k elements.*

for each process i **do**

Sorted access one depth in L_i, and sorted access T depths in parallel to $L_1, \cdots, L_{i-1}, L_{i+1}, \cdots, L_m$ in one super step. At each super step d (when Td objects have been accessed under sorted access in List $L_j, (j \neq i)$ and d objects have been accessed in List L_i) do:

- Maintain the bottom values $\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \cdots, \underline{x}_m^{(d)}$ encountered in each list.
- For every object R with discovered fields $S = S^{(d)}(R) \subseteq \{1, \cdots, m\}$, compute the $W^{(d)}(R) = W_S(R)$ and $B^{(d)}(R) = B_S(R)$. (For objects that have not been seen, these values are virtually computed as $W^{(d)}(R) = t(0, 0, \cdots, 0)$, and the best value $B^{(d)}(R) = t(\underline{x}_1, \cdots, \underline{x}_m)$, which is the threshold value.)
- Let $T_k^{(d)}$, the current top k list, contain the k objects with the largest $W^{(d)}$ values seen so far (and their grades); if two objects have the same $W^{(d)}$ value, then ties are broken using the $B^{(d)}$ values such that the object with the highest $B^{(d)}$ wins (and arbitrarily among objects that tie for the highest $B^{(d)}$ value). Let $M_k^{(d)}$ be the kth largest $W^{(d)}$ value in $T_k^{(d)}$.

Call an object R *viable* if $B^{(d)}(R) > M_k^{(d)}$. Once those two conditions are met in some processor (assumed P_t): (a) at lest k distinct objects have been seen and (b) there are no viable objects outside $T_k^{(d)}$, processor P_t broadcasts abort commands to eliminate other processes. And returns its local $T_k^{(d)}$ as result.

end for

Fig. 2. Parallel-NRA Algorithm

Proof: From PNRA algorithm, one process p_i broadcasts abort commands to others and return its local $T_k^{(d)}$ when both stop rules (a) and (b) are satisfied in its local space. According to Lemma 2, this local $T_k^{(d)}$ is the correct top-k elements set. ■

Applied to the example shown in Figure 1, our PNRA algorithm is processing like this. For processor P_2, stopping conditions can not be met until both lists reach depth n. For processor P_1, at super step $\lceil \frac{n}{T} \rceil$ (when n objects have been accessed under sorted access in L_2 and $\lceil \frac{n}{T} \rceil$ objects have been accessed in L_1) stopping conditions (a)(b) are met. Then P_2 broadcasts abort command to others and return its $T_k^{(d)}$ as the answer to top-1 query. So the cost of PNRA is $(n + \lceil \frac{n}{T} \rceil) * C_s$, which is less than cost of NRA.

C. Cost of Parallel-NRA Algorithm

Theorem 2. *If NRA algorithm halts at depth d in one top-k query, then the cost of Parallel-NRA algorithm is at most* $\max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\} * C_s$.

Proof: According to Lemma 2, let L_1 be the list which needs d sorted accesses to accomplish query, and L_m be the list which needs d_{min} accesses.
For processor P_m,

1) $Td_{min} \geq d$. When algorithm PNRA performs Td_{min} sorted accesses in lists $L_1, L_2, \cdots, L_{m-1}$ in parallel and d_{min} in L_m. $\vec{d} = (Td_{min}, Td_{min}, \cdots, Td_{min}, d_{min})$ is top-k sufficient, since d_{min} is the min value among all minimum components of vectors in S_d. So the cost of P_m is at most $(T(m-1)+1)d_{min} * C_s$.
2) If $Td_{min} < d$, then $\lceil \frac{d}{T} \rceil > d_{min}$. Depth vector $\vec{D} = (T\lceil \frac{d}{T} \rceil, \cdots, T\lceil \frac{d}{T} \rceil, \lceil \frac{d}{T} \rceil) \geq (d, d, \cdots, d, d_{min})$ because $T\lceil \frac{d}{T} \rceil \geq d$. Thus $\vec{D}$ is top-k sufficient. So the cost of P_m is at most $(T(m-1)+1)\lceil \frac{d}{T} \rceil * C_s$.

From subcase (1) and (2), $cost(P_m) \leq \max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\} * C_s$. Because the cost of Parallel-NRA algorithm is minimum among all processors costs, the cost of Parallel-NRA is no more than $\max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\} * C_s$. ■

Theorem 3. *PNRA algorithm is instance optimal.*

Proof: If algorithm NRA halts at depth d in top-k query, the cost of NRA is $dm * C_s$. The cost of PNRA is no more than $\max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\} * C_s$ from Theorem 2. NRA algorithm is proven to be instance optimal in [5] with a optimal ratio of m, we conclude that PNRA is instance optimal with ratio $\frac{1}{d}\max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\}$. ■

Theorem 4. *When NRA halts d depth in top-k query, if $d_{min} \leq \lceil \frac{d}{T} \rceil$ and $T \geq \frac{d+1}{d+1-m}$, the cost of PNRA algorithm is no more than NRA in this query.*

Proof: From Theorem 2, the cost of PNRA is no more than $\max\{(T(m-1)+1)d_{min},\ (T(m-1)+1)\lceil \frac{d}{T} \rceil\} * C_s$. When $d_{min} \leq \lceil \frac{d}{T} \rceil$, we get $cost(PNRA) \leq (T(m-1)+1)\lceil \frac{d}{T} \rceil * C_s$.

1) $\lceil \frac{d}{T} \rceil \times T = d$. $(m-1)d + \lceil \frac{d}{T} \rceil \leq md$, which leads to the cost of PNRA is no more than NRA.
2) $\lceil \frac{d}{T} \rceil \times T = d + 1$. From $T > \frac{d+1}{d+1-m}$, we get $mT + d + 1 < dT + T$. then $(d+1) + (d+1)(m-1)T < mdT$. That is $T\lceil \frac{d}{T} \rceil + T(m-1)\lceil \frac{d}{T} \rceil < md$. The cost of PNRA is no more than NRA.

Combining subcase (1) and (2), we get the conclusion. ■

D. Randomized Parallel-NRA Algorithm

Using random $rand(T)$ as stride of super step in PNRA algoithm shown in Figure 2, we get randomized Parallel-NRA algorithm(RPNRA) . If the random $rand(T)$ has a range $(1, \cdots, T_{max})$. It can be easily proven that RPNRA is correct and instance optimal, similar to Theorem 1 and Theorem 3 in Subsection III-C.

IV. Experiment Results

In this section, we present the experiment results of PNRA algorithm and NRA algorithm. Experiments are performed on both synthetic and real-word data. We used synthetic data in [9] and real-word data from [11]. Algorithms are implemented in C++ and MPI. We do experiments on Lenovo 1800 server in Supercomputing Center of USTC. The server contains 64 computing nodes. Each node contains two 2.33GHz Intel E5410 CPUs(four-cores, 64 bits) and 4GB memory [12].

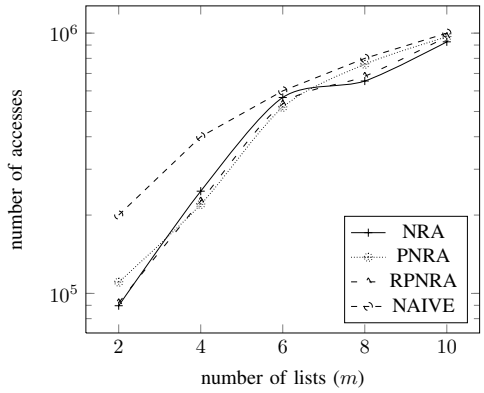

Fig. 3. Exponential Distribution Data

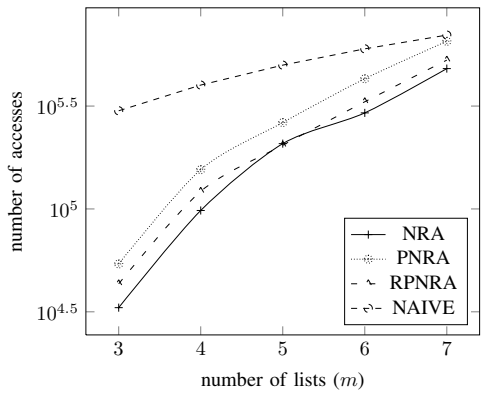

Fig. 4. Uniform Distribution Data

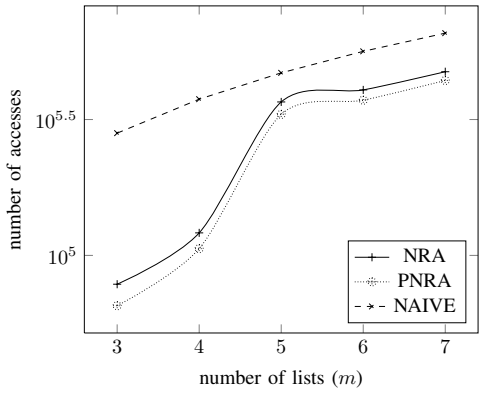

Fig. 5. Elnino Data Set

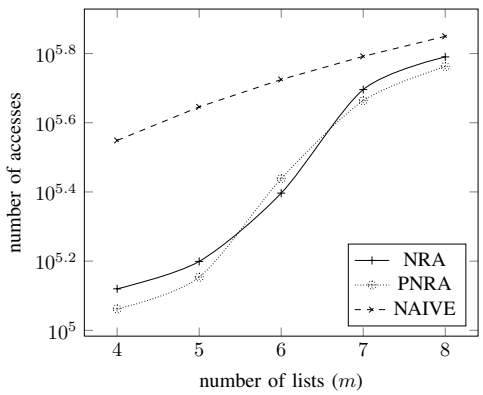

Fig. 6. IPUMS Census Database

A. Experiments on Synthetic data

We experimented two kinds of generated data, uniform distribution(UI) and exponential distribution(EXP). All generated values range from 0 to 1. The default parameters are $n = 100000$, $k = 20$, $T = 2$, $max\{rand(T)\} = 2$, aggregation function is sum.

It is shown that algorithm PNRA and RPNRA both outperform algorithm NRA on exponential distribution data in Figure 3. However on uniform distribution data, algorithm NRA costs less than our PNRA and RPNRA shown in Figure 4. That is because in uniform distribution components of necessary depth array are numerically close, d_{min} is more than $\lceil \frac{d}{T} \rceil$. The NAIVE algorithm [5] is worst in both Figure 3 and Figure 4.

B. Experiments on Real-Word Data

We also compared our algorithms to NRA and NAIVE algorithm on real data as an addition. The data sets used here are cited from [11]. Aggregation function is summation.

The first real-word data set is El Nino Data Set which was collected with the Tropical Atmosphere Ocean (TAO) array . The data set is mainly used to understand and predict seasonal-to-interannual climate variations originating in the tropics. We discarded those lists with missing values and took 7 lists as our testing set. All values are normalized by $x_{new} = \frac{x-min}{max-min}$, where min and max are the minimum and maximum values of corresponding list respectively. Number of objects is 93935.

The second data set is IPUMS Census Database Data Set, which contains unweighted PUMS census data from the Los Angeles and Long Beach areas for the years 1970, 1980, and 1990. All values are normalized by $x_{new} = \frac{x-min}{max-min}$. We chose 9 attributes to perform top-k queries. The set contains 88443 objects.

Figure 5 and 6 both show that our PNRA algorithm outperforms NRA on real-world datasets, and hence outperforms NAIVE algorithm. When compared to NRA algorithm, a benefit of up to 15.8% less accesses can be achieved by using PNRA on Elnino dataset. The benefit against NRA ranges from 6.25% to 14.3% by applying PNRA algorithm on IPUMS Census Database.

C. Discussion

The experiments on generated and real dataset show that our PNRA algorithm can outperform original NRA algorithm in many applications. PNRA is quite suitable for those queries that max depth d is numerically much lager than min depth d_{min} of necessary depth vector. However since d and d_{min} are numerically close in uniform distribution data-set, NRA

algorithm is better than PNRA in that case. Experiments on synthetic data also show that RPNRA algorithm is more "moderate" than PNRA. When PNRA is better than NRA, RPNRA is also better than NRA but worse than PNRA, vice verse.

V. Conclusions

In this paper we proposed parallel algorithms (PNRA and RPNRA) for top-k queries on parallel computers. These algorithm use symmetry breaking strategy to generate depth array based on the observation that NRA algorithm accesses more objects in these queries with unbalanced sufficient and necessary depth array. We prove the correctness and instance optimality of PNRA. We also analyze the conditions on which PNRA cost less than NRA and give an upper bound of the cost of PNRA algorithm. Experiment results on both synthetic and real data show that PNRA can outperform original NRA algorithm in many applications.

Our paper considers mainly on the queries requiring an unsymmetrical depth array. We studied the situation that d_{min} is far less than other depths. On the other side, when d is far lager than other depths, how to modify NRA algorithm to gain low cost is a issue of our future research. And we are also interested in dividing lists to more subsets and apply different strides of super step to different subset.

Acknowledgement

This work is supported by the Fundamental Research Funds for the Central Universities, National Natural Science Foundation of China (No.60873210) and National High Technology Research and Development Program of China (No.2009AA01A134). This work is also supported by Anhui Provincial Natural Science Foundation (No.090412064).

References

[1] K. Asanovic, et al, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Tech. Rep. UCB/EECS-2006-183, Dec. 2006.

[2] X. H. Long and T. Suel, "Three-level caching for efficient query processing in large web search engines," *World Wide Web*, vol. 9, pp. 369-395, Dec. 2006.

[3] L. Getoor and C. P. Diehl, "Link mining: a survey," *ACM SIGKDD Explorations Newsletter*, vol. 7, pp. 3-12, Dec. 2005.

[4] S. Nepal and M. V. Ramakrishna, "Query processing issues in image (multimedia) databases," in *Proc. ICDE*, 1999, pp. 22-29.

[5] R. Fagin, A. Lotem and M. Naor, "Optimal aggregation algorithms for middleware," in *Proc. PODS*, 2001, pp. 102-113.

[6] U. Güntzer, W.T. Balke, and W. KieBing, "Towards efficient multi-feature queries in heterogeneous environments," In *Proc. ITCC*, 2001, pp. 622-628.

[7] R. Fagin, "Combining fuzzy information: an overview," *ACM SIGMOD Record*, vol. 31, pp. 109-118, Jun. 2002.

[8] N. Mamoulis, M. L. Lung Yiu, K. H. Cheng, and D. W. Cheung, "Efficient top-k aggregation of ranked inputs," *ACM TODS*, vol. 32, Aug. 2007.

[9] J. Yuan, G. Z. Sun, G. L. Chen, and Z. Liu, "Selective-NRA Algorithms for Top-k Queries," in *Proc. APWeb/WAIM*, 2009, pp. 15-26.

[10] R. Fagin, "Combining fuzzy information from multiple systems," in *Proc. PODS*, 1996, pp. 216-226.

[11] UC Irvine Machine Learning Repository. [Online]. Available: http://archive.ics.uci.edu/ml

[12] Supercomputing Center of USTC homepage. [Online]. Available: http://scc.ustc.edu.cn