TOP K QUERY PROCESSING IN DISTRIBUTED DATABASE

by

AMRITA TAMRAKAR

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2007

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Gautam Das for his guidance and constant support during my research work. This work helped me understand in depth about the ranking in Database and information retrieval. Also I thank Dr. Das for involving me in the Query Reformulation project through which I got to learn all the research work going on in that area.

Also I thank my committee members Dr. Leonidas Fegaras and Dr. Nan Zhang for their comments and suggestions.

I thank Lekhendro Lisam for his help throughout my thesis and Kavya Reddy Musani for her strong belief in me.

Last but not the least, I would like to thank all my friends at UTA in helping me concentrate in the research work and always motivating me to march ahead in face of obstacles. I would like to thank my parents, brothers and sister for their constant support in all I have done.

July 23, 2007

ABSTRACT

TOP K QUERY PROCESSING IN DISTRIBUTED DATABASE

Publication No. _____

Amrita Tamrakar, M.S.

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Gautam Das

Today's data is rarely stored in centralized location due to the enormous amount of information that needs to be stored and also to increase reliability, availability and performance of the system. Same data is stored in different format into different company's database as well as they may be partitioned or replicated. We consider various scenarios of distributed database such as horizontal, vertical fragmentation and attribute overlapping. Allowing access to integrated information from these multiple datasets can provide accurate and wholesome information to the end-user. We research on efficient querying to these distributed databases to get top k elements matching the ranking order provided by the user. We also discuss ways of using the top k algorithm and their limitations to our problem. We propose four different algorithms based on NRA algorithm to solve this problem efficiently and compare and contrast these

methods. Once the combination of data sources has been identified, we use our algorithms to get the top elements from these data source combination, process them to get the top k elements according to the user's ranking function.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Top k query processing is among the most researched techniques for ranking in huge databases and information retrieval. The web and the data storage is increasingly moving away from centralized system to distributed system. The web is a huge database repository which stores data in various locations. Similarly the relational databases have also come along way from just a simple storage to Data Warehousing, Data Mining, Replication and Integration. They are managed by independent entity so they have different schema and different data. For e.g. In a real world scenario, every university has it's own database and even more than one database, similarly every hospital has one or more databases. If an insurance company wants to find out the top k students who have been in a car-accident, so as to research on its insurance policy for such group, it needs to combine university, hospital and police database which are all independent entity. Another example would be to research on the health risk of diabetes patient with high blood pressure level. The research has to cover multiple hospital databases which are autonomous in themselves.

In the example shown in the following figure, an insurance company wants to find out the top k students who have been hurt in multiple car accidents so that they can come up with new policies and market it to these groups. In order to get the results, they

need to get the students information based on their academic grade, no. of traffic violation citations and their hospital visits. That means assimilating the information present in three different databases and ranking them according to the attributes distributed across them.

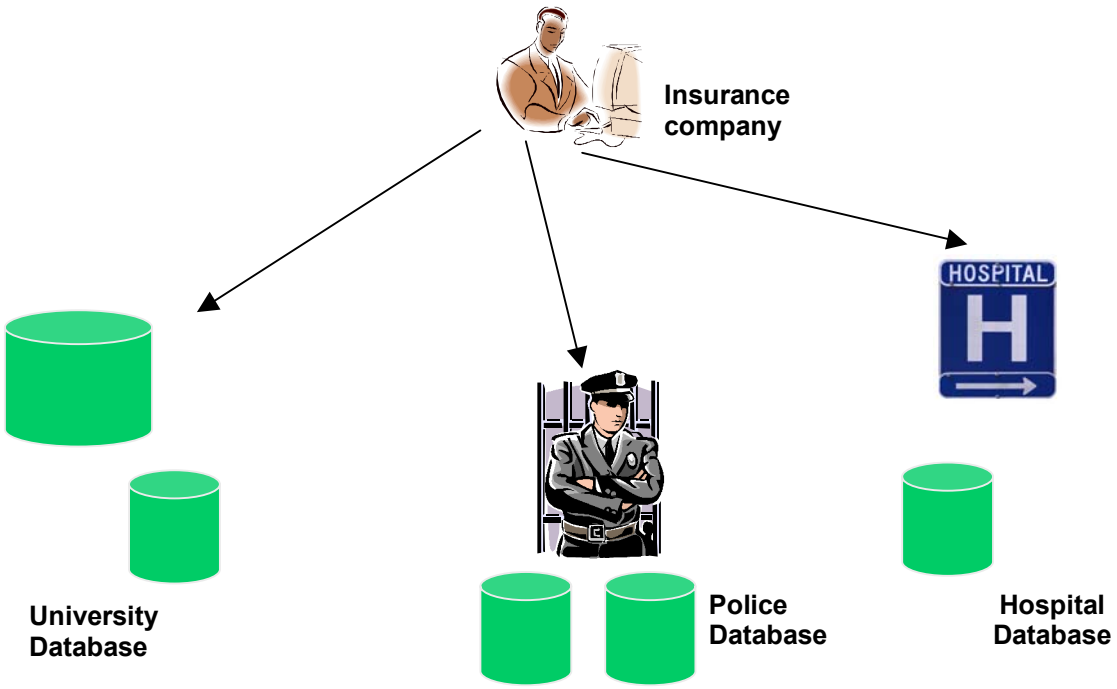

Fig 1 Distributed database depicting data from various organizations

Even though the above picture depicts some individual organization, they may have fully or partially replicated database. Each of these databases, if seen in detail reveals the disparity of having different attributes and different records amongst themselves. For e.g. university database, police database and hospital database contains following tables.

ssn	Name	Dept	Major	Grade
xxx	Joe M.	CSE	IT	A
yyy	Mary J.	EE	Nano	B

Ssn	Name	Citations	Vin	LicenseNo	AccidentType
xxx	Joe M	3	1	123456	Car
yyy	Mary J.	2	5	457899	Car

ssn	Name	AccidentType	No.of visits	Diagnosis
xxx	Joe M.	Car	10	Fracture etc

ssn	Name	No. of visits
yyy	Mary J.	20

ssn	Accident type	Diagnosis
yyy	Car	clotting

Figure 2 Sample tables of different databases of university, police and hospital departments respectively.

Even each of the databases may be partitioned and kept into different locations. For e.g. patient records may be collected from hospitals around the universities. Similarly another example can be taken from a diabetic research center which wants to find out the relation between diabetes and high blood pressure. Hence it wants to find out the top k patients for research purpose. The research center would have to take

many hospitals and diabetic centers into account. Each of these is maintained individually and hence as seen from following figure, their attributes and records differ.

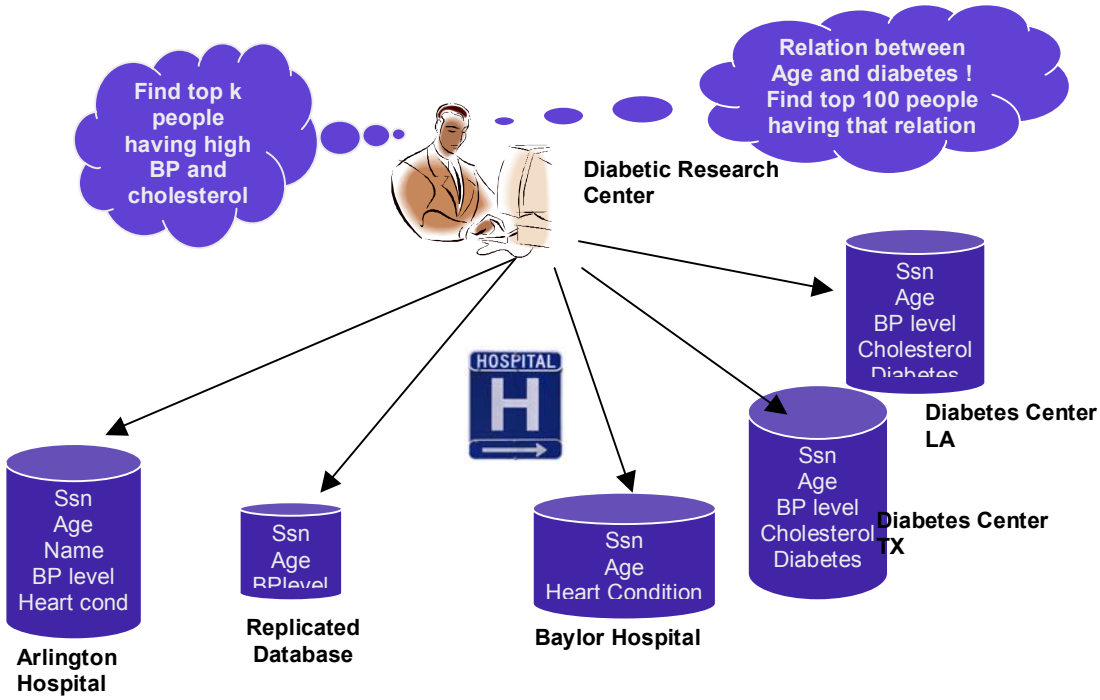

Figure 3 Ranking in Distributed Hospital databases

In our thesis, we try to focus on solving the top k query processing in such a distributed environment. For simplicity we assume that the union of all these data sources would give us a whole gigantic database in its full form. For eg. Student A has attribute values in University database, hospital database and police database for the researcher to give the ranking function comprising of academic grade, type of surgery and no of accident. So if we combine all the three databases, it will contain all the attribute values for Student A. We assume that each of the databases has sorted list for

each of their attributes as well as they are capable of producing the sorted lists of attribute combinations as per the query request. We consider the distributed database as managed by individual entity and hence are independent in their schema and data management with each other. Since they are autonomous and distributed, we assume that random accesses are not allowed and only sequential accesses are allowed due to the cost factor as random access is cost higher than sequential access. Random access denotes the querying of database using the unique key while sequential access means getting the results one after another as output by the database.

These database can be considered as hybrid fragmentation of the gigantic database as we assume that when all the databases are combined, they give us the complete database i.e. each tuple element in the combined database contains every attribute. For e.g. we don't consider the following types of database partitions as depicted in the figure 2. Here certain tuples in D1 doesn't contain the values of attributes A5 and A6. Similarly certain tuples in D2 doesn't contain the values of attributes A1 and A2.

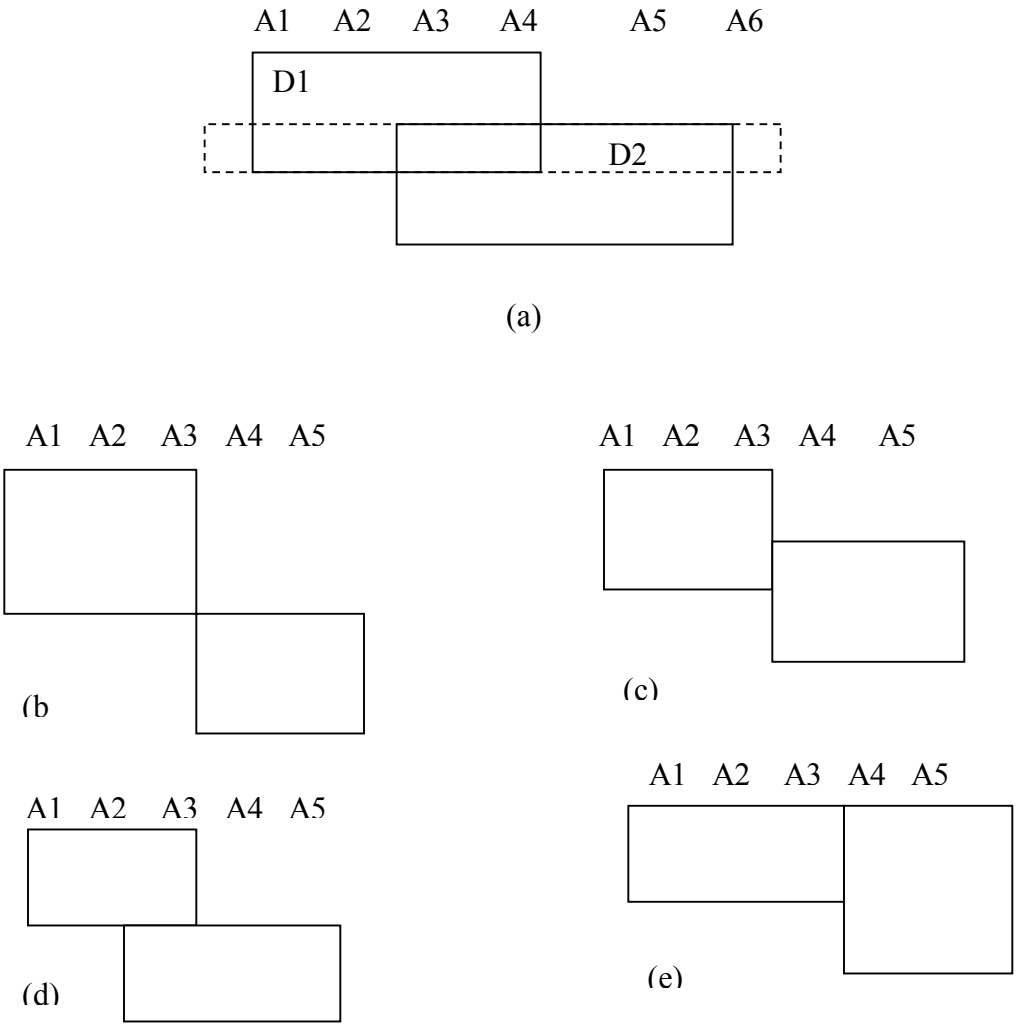

Fig 4: Some more databases with incomplete attributes which we are not considering due to the complexity involved.(a)Hybrid fragmentation with unknown attributes and also overlapping in records and attributes (b) Completely unknown partitions for all records (c) Partial records have all known attribute values (d) Only few attributes known to all records (e) few records have some unknown attributes

In our thesis we aim at providing a solution for ranking the results according to monotonic aggregation function on the distributed database so as to avoid the total scanning of all the databases.

An aggregation function f is said to be monotonic when the $f(x_i, x_j \ldots x_n) <= f(x_{i+1}, x_{j+1}, \ldots x_{n+1})$ is true where x_i are the attributes for every i.

We assume that each of the databases allows sequential access i.e. gives the output for each of its own attribute in the sorted order as a user demands. But our query requires that the result be ranked in the combination of these attributes. To help solve this problem without scanning the whole databases, we need to apply the top k selection algorithms. But due to the unique problem presented by the nature of distributed database, we cannot apply the top k algorithms as it is. Hence, we have proposed four different variations of the top k algorithm which will take advantage of the combined and the overlapped attributes present in different database in order to make the scanning stop as soon as possible. With the help of these algorithms we focus on increasing the efficiency for querying both in terms of access cost by round trip to the database and time reduction for termination.

CHAPTER 2

RELATED WORK

The basic and naïve approach of finding out the top elements by scanning all the database was almost revolutionized by the paper [20] published by Fagin et al and others [4, 5]. Among the many top k popular algorithms, TA has been the most researched and used in many variations throughout numerous other researches [6, 7, 8, and 9] on Top k query processing. TA is an instance optimal algorithm and uses bounded buffer to get the ranked results from a database limiting the scan depth on the lists in order to terminate the algorithm as soon as possible.

The NRA algorithm is one of the variations of TA which uses similar method as TA but doesn't allow random access and works only with the sorted access. Hence NRA is time consuming and requires unbounded buffer size unlike TA. But it is useful when scenarios where random access is quite expensive or impossible is taken into consideration. For the rest of our thesis we concentrate on the NRA or TA-Sorted algorithm.

Most of the papers on Top k Selections are based on either the web based information repositories or the lists where only one list per repository is involved. There is not much of work done on the distributed database or hidden databases in the web, which usually have duplicate attributes and duplicate records amid them.

Among the top algorithm discussed in distributed environment is TPUT algorithm [10]. It uses the TA algorithm and gets the results in a batch. There are three phases in the algorithm. Fetching the k best entries (docid, score) from each of the peers and compute the top k worst score 2) query each peer to send all tuples having score>=top k worst score/no of attributes 3) do RA to fetch unseen attributes to compute the actual score. Another algorithm KLEE [14] discuss about the information retrieval from distributed peers using the modified version of TPUT that uses NRA method. It uses bloom filters and histogram information to trim down the candidates in the second step of retrieval from the peers. Similarly [7] paper discusses the ranking of the tuples using modified TPUT but unlike KLEE which is based on approximate results, it give absolute ranking and uses RA method and uses non-uniform threshold across the peers. All of these works are focused on lists which are unique per peer.

Another work [2] related to distributed environment is done as early as in 2002. In this paper, it considers a web search engine which needs to get top k result from both the Random and Sequential access sources but it tries to minimize the random access. It performs restaurant ranking using three different autonomous web accessible databases with different interfaces. It assumes that one of the sorted lists provides the sequential access and it contains all the record ids while other lists can perform only random access. In our case, one of our main limitations is that a single list may not contain all the records. Another important paper [11] has shown similar work in self-tuning query expansion which uses nested top k and incremental merging of inverted list. It uses the TA algorithm and also probabilistic estimators. Basically it involves candidate lists

which are list of the expansion terms with their frequency of occurrence as their score. The incremental merging combines these candidates' lists incrementally using score function which is the combination of similarity function and the score for each element and continues until the threshold (top k min) value is reached.

Another area of top k algorithm is the join algorithms. [21] Discusses of TJA algorithm which uses the peers as nodes which send their results to its parent. The intermediate nodes perform the merging of all the received lists and send the top k to the parents. So it acts like a tree which sends the results in batches from root to the parents meanwhile the threshold information travels from top to bottom. Similarly [17] uses the TopX search engine which uses hash joins for merging the partial scores and uses efficient candidate queuing and index access scheduling, incremental merging of inverted lists with potential expansion terms and forms virtual joins. It uses nested top k query with tree structure for phrase matching. It reads complete blocks instead of single element termed sorted block-scans but is capable of random access when needed. RankSQL [12] discusses on changing the query algebra by interleaving the ranking in the join conditions in the database itself instead of first querying and then ranking the results. It advocates fundamental support of ranking in database itself instead of middleware. Unlike our problem, these algorithms consider one attribute per peer or one sorted list per peer. Though, we can also include the join conditions involving the tables distributed across various locations as our future scope. Currently we focus mostly on the selection condition involving certain attributes scattered in various databases identified by their unique tid.

The more recent work on top k using preprocessed views [1] brings into play the linear programming methods to terminate the algorithm efficiently. This is very near to what our LPNRA algorithm performs during its threshold computation. It selects a set of views available after determining the views which will give the top element faster. It uses histogram information and convolution of histograms to determine them quickly. Each of these views has different scoring functions and the algorithm gets the threshold value or maximum of the unseen tuples using the linear programming. It performs Random Access in the main relation table to get the rest of unseen attribute values of the seen tuples in the top k buffer. In our problem, we assume that we do not have a complete relation which has all the attributes for all the records.

CHAPTER 3

ISSUES IN DISTRIBUTED DATABASE

3.1 Hybrid Fragmentation

There are three type of fragmentation usually associated with databases. They are as follows:

i) Vertical Fragmentation

ii) Horizontal Fragmentation

iii) Hybrid Fragmentation

Vertical Fragmentation deals with the attributes of the database. Assume a table consisting of $a_1 \ldots a_n$ attributes and m tuples. When the table is partitioned into two tables with $a_1 \ldots a_k$ attributes in one table and $a_{k+1} \ldots a_n$ in another, each table containing m tuples, this type of fragmentation is called pure vertical fragmentation.

Horizontal fragmentation deals with the rows of database. Assume a table with millions of record and we need to keep the records in separate disks to have a faster retrieval, then table can be partitioned horizontally i.e. some records are kept in one table while others in next table. The schema of the tables will remain the same.

Hybrid fragmentation involves both these types of partitioning. In this case, the tables may have different schema, different data or same schema and different records or same records. The fig. 3 below depicts a hybrid fragmented database, we can see that tuple with id 25 has its attribute values stored in five different databases D1, D2, D3,

D5 and D6. Also all the tuples don't reside in one database or rather no database may contain the complete lists of tuples.

3.2 Overlapping Attributes and Duplicate records

This is one of the important situations considered in this research thesis. Most of the on-going researches on ranking have not considered overlapping attributes but this is a very common occurrence with the distributed environment and practical problem. Same attributes may be stored in various databases since they are essential to almost all of them meanwhile some attributes may be specific to some databases. This overlapping of attributes makes it very difficult to find the efficient ranking solution in the distributed databases.

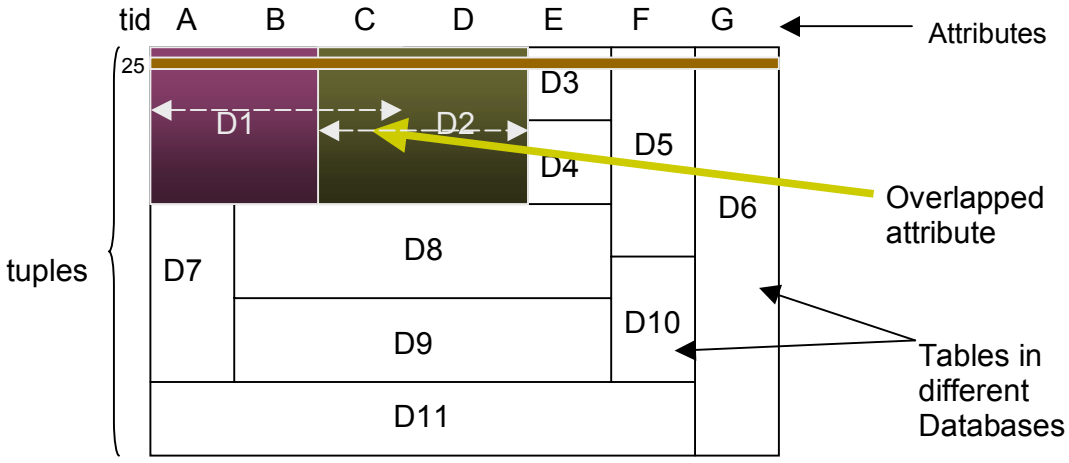

Fig 5 Vertical overlapping of attributes in databases

Similarly there may be duplicity in records among all databases and hence our solution needs to neglect the duplicate data from the ranking procedure.

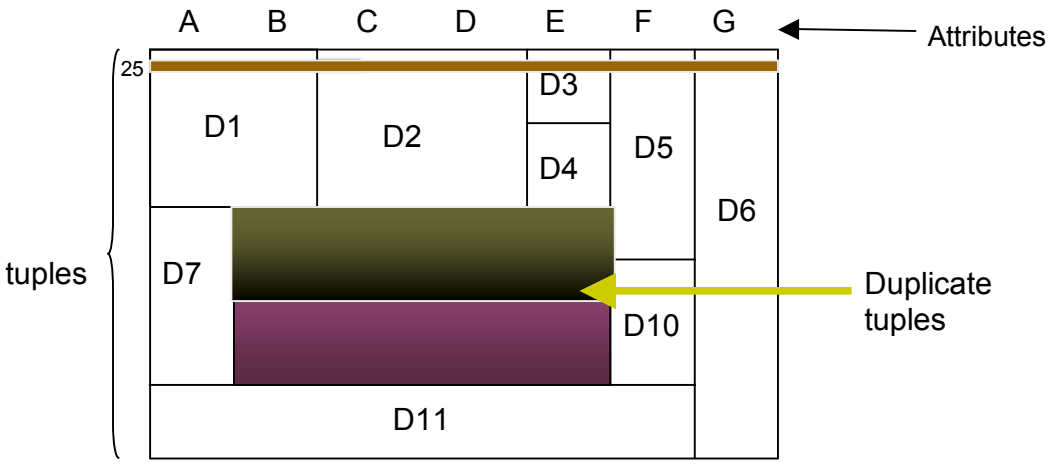

Fig 6 Horizontal overlapping of tuples with partial attributes in databases

3.3 Issues not considered in the thesis

3.3.1 Schema Matching

As seen in the above example of distributed database, the schema of the autonomous database is determined individually. This is a basic problem in database related applications such as Data warehousing, Data integration and Semantic query processing. For e.g. If the database D1 has attributes “age” and “name” while database D2 has attributes “year” and “name”, the two attributes “age” and “year” may denote same attribute but it is not obvious from the schema. Similarly the attribute “name” in these two databases may denote two different types of name i.e. they are not the same attributes even if they have same attribute name. The domain of schema matching has significantly grown from manual schema matching to automated schema matching with many limitations [17]. But it is a different domain in itself which we are not focusing in our research.

3.3.2 Value Matching

Similar to the schema matching, the attribute values of the same tuple may be different in different database due to misinterpretation, error or conformance rules. Finding these is very tough job and we consider in our research that they are all same wherever they occur. So if a tuple with tid t_i has attribute A value a_i in Database D1 then we assume that tuple with tid t_i in any other databases will also have value a_i for it's A attribute.

CHAPTER 4

RANKING ALGORITHMS

4.1 Ranking in Extreme Cases

In this thesis, we consider two extreme cases for distributed database. They are the pure horizontal and vertical fragmentations. For these types of partitions, we can easily provide a ranking solution.

If we have a pure horizontal fragmentation, we can get the top most elements from each database and then compare the top elements and then output the highest.

For a pure vertical fragmentation, we cannot apply this method because of the missing attributes. The solution is provided by basic NRA algorithm.

4.1.1. NRA Algorithm

According to Fagin et al, NRA is performed on situations where random accesses are not possible and we are given m sorted lists where m is the number of attributes. This can be considered pure vertical fragmentation. The algorithm is as follows:

1) Let $Score_Q$ () be the monotonous aggregation ranking function.

2) Do sorted access in parallel to each of the m sorted lists and loop until the top object is found. Maintain the top k min found in the lists.

3) For each tuple encountered, compute it's worst score based on the $Score_Q$ function on seen values and unseen values as 0, compute it's best score similarly but take maximum value seen till now for the unseen attributes.

4) Let T_k be the current top k list, containing the k objects with highest worst scores seen so far; ties are broken arbitrarily if the scores are same. Let top k min be the k^{th} largest worst score tuples.

5) Call an object viable if Best Score of the tuples is greater than top k min. Halt when at least k distinct objects have been seen and there are no viable objects left outside T_k.

4.2 Ranking in Intermediate Cases

In this research, we consider intermediate case as the one when we have hybrid fragmentation. As we had discussed earlier, NRA works on two or more different attributes which are sorted in descending order. In our case, we have various problems which cannot be solved by simple NRA algorithm as it exists. The main problem is database independence and although we assume that the data is consistent through out the system, the design of database schema and the records in the database is solely maintained by the individual organizations. These will very likely result in same attributes across the different databases. These databases may share the same records or they may be disjoint or may have certain percentage of intersection.

One of the main interests in our research is to show that we can exploit the combined attribute ranking as needed by the scoring function. For e.g. If the query desires A+B+C as the scoring function and if one of the database has A and B, then instead of getting A and B as separate attributes sorted individually, we can gain more time and access cost by getting them as A+B sorted. Since the results will be already sorted according to our requirements, we are certain to gain in our efficiency. While this seems to be a very thoughtful process, it becomes more interesting when we see the practical sides of the distributed system which consists of many overlapping in the attributes. Hence the intermediate case involves the following:

i) Overlapping Attributes

ii) Hybrid Fragmentation

4.2.1 Difficulties

Let us go a bit into detail of how this simple practical problem renders it very difficult to find out the solution using simple NRA.

In order to apply the top k algorithms we need to find out the three important variables. They are the threshold value per parallel access, Best Score and Worst score value per tuple and the top k min value. Among these, the worst score and top k min is easy to find as they are the aggregation of the seen values. But computing the threshold value and the best score for each tuples gets tricky and complex due to the lists that are ordered in the combined attribute scores rather than individual ones.

Suppose we have a database which has A, B and C as attributes and another database which has B and D as the attributes. If our scoring function is the aggregation of A, B, C and D, and the tuple t_i and tuple t_k are seen from the current access from D1 and D2. What will be the threshold value? We can take Score (t_i) for A+B+C but for D value we cannot take the D value of tuple t_k as the highest. That is because the t_k has the Score (t_k) but it's D value may be less than any unseen tuples whose combined score Score(t_{unseen}) is less than Score(t_k) but D value of t_{unseen} > D value of t_k . So determining the maximum D value gets harder.

We propose four different algorithms to solve this type of ranking. They are as follows:

1) Individual Attributes Ranking (INRA)
2) Combined Attributes Ranking (CNRA)
3) Max NRA
4) LPNRA

We use merge-sort operation in all of these algorithms.

4.2.2 Merge-Sort Operation

This operation gets one object at a time from the input queues. It operates as follows:

Merge-sort

Let *qi, qj....qn* be the sorted attribute lists of the same attributes located in different databases.

while (true)

if the top buffer is empty

{Do parallel access to each of the list

Do merge-sort opertion in the buffer}

else

{Perform sequential access to the database (Dt)

Do merge-sort operation in the buffer}

Record the database with top object (Dt)

Output the top object

4.2.3 Individual Attributes Ranking (INRA)

As in the original NRA algorithm, here we use the basic individual attributes from each database. To solve the problem of overlapping attributes, we do the algorithm in three steps:

1) Get the database schema for the all of the databases.

2) Create a merge-sort tree for all the attributes that are overlapping, thus eliminating the duplicate attributes for the final NRA. Access the output from this tree for the attribute.

3) Perform NRA over all attributes.

Fig 7 Individual NRA using total vertical partitioning

In the fig no 5, Database D1 will provide individually sorted list A and List B; similarly Database D2 will provide sorted lists C and D. The list with common attributes are merged first and sorted, then NRA is performed on the output of these sorted buffer list. The following figure is another example of how this occurs.

The improvement here from the original NRA is that instead of performing the access for the objects in sequential way for the duplicate attributes, we are using a merge-sort operation (section 4.2.2.2) which is more efficient. Nonetheless, this is the basic approach we are undertaking for solving the problem.

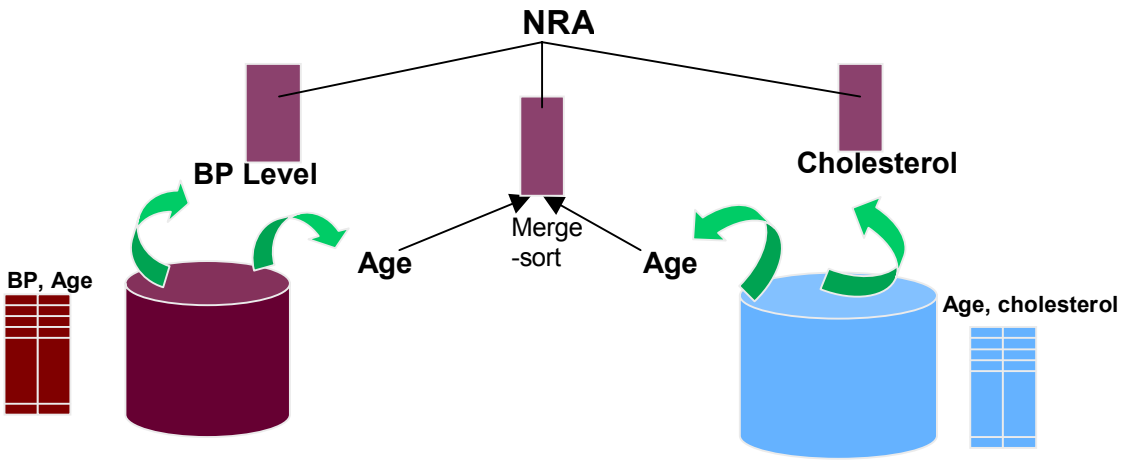

Fig 8 Individual NRA using merge-sort and basic NRA

In the above algorithm, we take advantage of the fact that they are the same attributes which are already sorted. So once we have seen one object, we know that the next object will have lesser score than the currently seen one.

In the first access, we do parallel access to all the databases, perform merge operation in a buffer, sort it and output the topmost object. Suppose that object was seen

in D2, and then we need to access the next object from D2 only, since we already know the top objects of all other databases which were lower than the top object of D2. Hence we gain the access cost from all the other databases.

4.2.4 Combined Attributes Ranking (CNRA)

This approach is one of the main causes of our research. We have two approaches for this system of solving the problem.

In the first approach, we try to solve the problem using the advantage of already ranked attributes by various databases.

As opposed to earlier approach, when we take individual attributes and perform the top k processing, we are not performing duplicate computations. If the attributes can already be ranked in the individual databases, then we are sure to get the candidates which are more likely to be in the top most categories. for e.g. Given A, B and C attributes, the resulting tuples from A+B ranked lists are more likely to be in the list than the tuples from A or B alone.

Hence we try to match up more combined attribute list as possible. The main obstacles here are same as in the pervious approach. We have vertical as well as horizontal fragmentation.

How do we get the top element from two databases which share common attributes?

The difficulty here is finding of the threshold values and the best score for each object. By threshold value in NRA or any other top k algorithm, we mean a value which

is the highest that any unseen object can get till now. This value is calculated by applying the scoring function on the top value from the most recent parallel access to all the lists.

For e.g. Attribute Ai, Aj .. An have values $v_i, v_j, .. v_n$ after k^{th} parallel access, and if the scoring function is the summation of all the attribute values, then the threshold will be $SUM(v_i, .. v_n)$ even if the object ids are different for these values.

Now when we have overlapping attributes, say A1, ... An then, how to calculate the threshold value? It cannot be the summation of all these attribute values. The point to remember here is that the value per access for the same attribute from two databases may differ as they are getting two different records. Another important thing to consider is that we are accessing the database according to the combined attribute sets. This does not guarantee that each individual attributes are highest seen till now. i.e. If $v_{11} \ldots v_{n1}$ are the values of these attributes for Database D1 in kth access, then $Score_Q(t_k) > Score_Q(t_{k+1})$ but v_{1k} may not be greater than $v_{1(k+1)}$. Since we do not know the individual best of the attributes, we cannot calculate the threshold and the best score for each object.

As a solution to this deadlock, we use the combined Algorithm which takes two approaches.

In the first variation of the algorithm, we take only those combined attributes which are common to both the databases. For e.g. if Database D1 has A, B and C attributes and Database D2 has B, C and D attributes, then we take, A, B+C from D1 and B+C, D from D2. We perform a merge-sort operation on D1 and D2 for B+C

attributes as the first step and then perform the NRA over A, B+C result and D. We can calculate the threshold as A + max (B+C) + D. Since (B+C) acts as a separate attribute say ‘X ‘for each database here, we don't have to calculate the individual attributes separately. Similarly to calculate the best score for each object we can get the total score of the attributes and get the maximum among them.

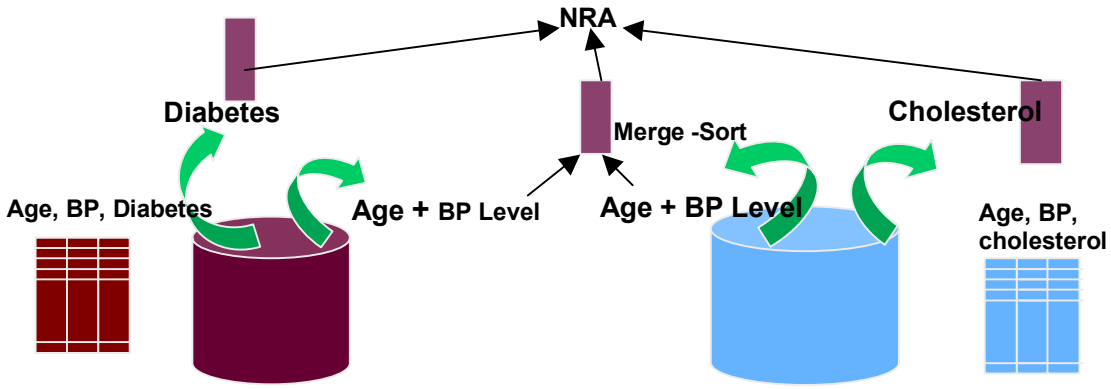

Fig 9 Combined NRA with taking inputs from overlapped attributes

CNRA Algorithm

- Find the combined attributes (eg. A+B) that are present in all database
- Query the sorted list for these same set of combined attributes
- Merge and sort the results to give the top output.
- Compute NRA over these outputs with dissimilar attributes.
- Loop until top k output

In the above figure, the Age and BP level are two attributes which are common to both the database and hence we query the sorted lists for these combined attribute and perform merge-sort operation. Then we compute NRA algorithm on the output with other dissimilar attributes.

Another approach is that we do sequential access on the database having the maximum combined attribute. Then access other database for the remaining attributes. For e.g. in the above database D1 and D2, we access D1 for A+B+C and D2 for D alone.

- Query the sorted list for the combined attributes from all database
- Check if the attribute has been already present in another database
- Merge and sort the results to give the top output.
- Compute NRA over these outputs with dissimilar attributes.
- Loop until top k output

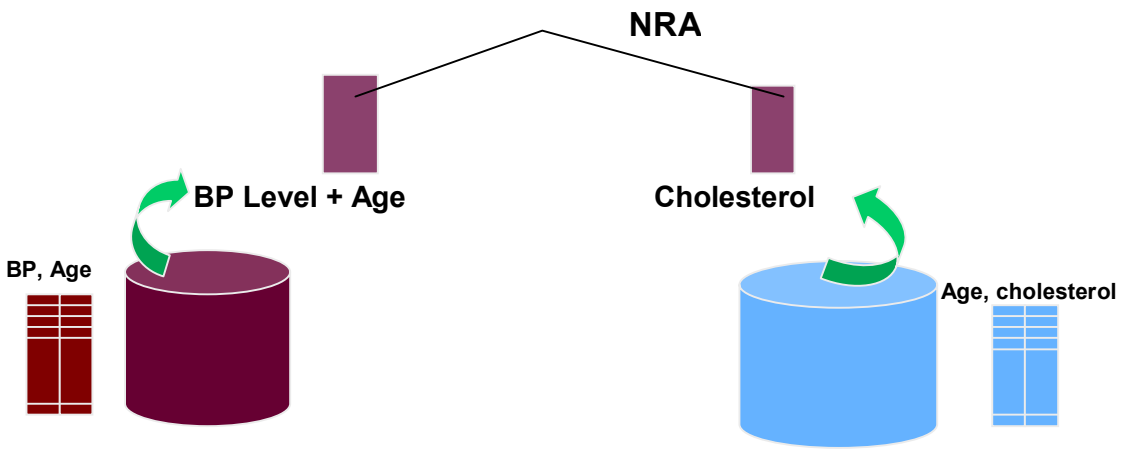

Fig 10 Combined NRA without using the overlapped attributes

There are various forms of database schema, which may require some extra work to find out the most efficient way applicable to them.

4.2.5 MaxNRA

This is the most flexible yet least efficient method and hence least preferable method. In this method, we assume that the unknown values for the unseen attributes are 1 i.e. maximum possible value.

MaxNRA Algorithm

- Query the sorted list all attributes from all database
- Compute the bestscore and threshold for tuple with unseen attribute value as 1
- Perform NRA over all the sorted lists.
- Loop until top k output

This causes more flexibility as regardless of database schema, we can do parallel access in all databases and get the results in buffer meanwhile inputting 1 for all unseen attribute values per object. The threshold can also be calculated in the same way. For e.g. If A+B = 1.4 from Database D1 and B+C=1.6 from Database D2, we can say that the max A+B+C for D1 can get 1.4+1=2.4 while for D2, it can be 1.6+1=2.6. Hence the threshold can be max (2.4, 2.6). Similarly the best score will be 2.4 for D1 and 2.6 for D2.

The inefficiency of this solution is that the stopping condition, which is threshold < top k min value. The worst Score value i.e. top k min will be quite small in

comparison to the threshold value which takes maximum of all. Until the unseen value is seen, it assumes the score of 1 and hence doesn't decrease. This causes the maximum buffer space and increased number of access to find out the top values. Hence this procedure may not be suitable for the distributed natured system.

4.2.6 LPNRA

This is among the important approach we have taken to resolve the problem we face when we take combined attributes. So it actually tries to minimize the number of accesses than the 2nd and 3rd approaches. This is the strictest algorithm as it demands that there be no horizontal fragmentation before the final NRA.

It takes the combined attributes from all the databases. The results from each database are ranked accordingly and we try to get as many combined attributes as possible. Now when we need to find the threshold value, we can use the linear programming methods to get the maximum value.

4.2.6.1 Linear Programming

LP method is mostly used in solving the complex business and statistic problems. In situations when we need to maximize profit given some constraints, we can convert the problem into linear equations and use various methods like simplex method, ellipsoid methods and interior-point techniques to get the maximum or minimum value.

A linear function to be maximized

e.g. maximize c1x1 + c2x2

Problem constraints of the following form

e.g. a11x1 + a12x2 <=b1

a12x1 + a22x2 <=b2

Non-negative variables

e.g. x1>=0, x2>=0

For e.g. a typical linear programming problem would be as following:

Given wheat (2$/lb) and corn (3$/lb), we need to prepare bread dough with 50$ so that we get the maximum profit. Each dough will weigh 1lb. What can be the maximum profit?

Let x and y be the amount of wheat and corn. Maximize c1x + c2y where c1 and c2 are the cost profit per unit of dough produced.

Constraints will be 2x +3y < =50, x+y <= 1.

In our case, we have the following cost function. Assume we have attributes A,B,C and D and database D1 has A,B and C attributes while D2 has A, C and D attributes. tot_i and tot_j are two scores from D1 and D2. Converting to linear problem :

Maximize A+B+C+D

Subject to

$a_1 + b_1 + c_1 <= tot_1$

$a_2 + c_2 + d_2 <= tot_2$

$0 =< a_i, b_i, c_i <= 1 \quad i = \{1...n\}$

Cost function for a linear program may be maximization as well as minimization. In our case, we will maximize the cost function. Similarly the constraints can be of various types such as greater than inequality, less than inequality and equals

to. In our case, the total score is always less than inequality since the tuples cannot get more score than presently seen scores since they are already sorted in descending order. Graphical depiction can help understand Linear programming solution and also help solve the problem.

In the following picture, the intersection of the two constraints depicts all the points that can be the solution satisfying the constraints, hence called feasible region. Since we need to maximize our solution, the vertices of the feasible region will give the solution to this problem

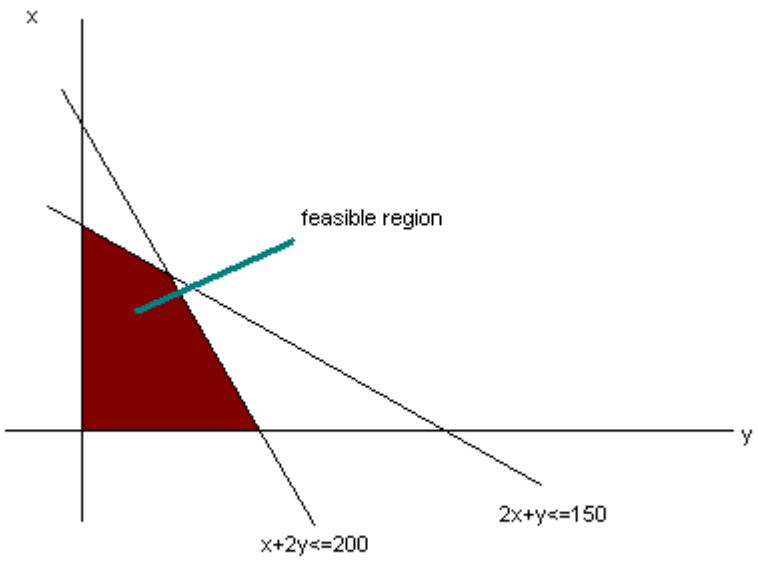

Fig 11 Feasible region for two linear equations

4.2.6.2 How does LPNRA works?

In case of our algorithm, we can get the threshold value for each access parallel access by converting it to a linear program. For e.g.

If Database D1 gives $a_i + b_i + c_i$ with total score as 2.9 and Database D2 gives $a_i + c_i + d_i$ with total score 2.3. To find out the threshold value or the unseen maximum for the upcoming tuples, the cost function will be to maximize $(a_i + b_i + c_i + d_i)$

Constraints will be $a_i + b_i + c_i <= 2.9$

$$a_i + c_i + d_i <= 2.3$$

Subject to $0 <= a_i, b_i, c_i >= 1$

Similarly, to calculate the best score for each object seen, we apply the same technique. We get the maximum seen from other databases till now and subtract the overlapped attribute we already have. For e.g. we have database D1 giving the tuples in the ranked order of A+B+C, while the other database have the results coming in B+C+D order. Suppose tuples t_1 ($a_1+b_1+c_1=score_1$) is from database D1 and t_2 ($a_2+b_2+d_2=score_2$) from Database D2. Now to get the best score for tuple t_1, we need to find max d_1, which we can get by subtracting b_1+c_1 from $score_2$ seen in database D2. We cannot get the d_2 value directly as the maximum d_1 can get, although the tuple t_2 has the maximum score seen till now , it is the total combined score of B+C+D attributes till now hence as discussed above, doesn't mean that the individual D attribute is also the maximum seen till now. Hence if we subtract the overlapped attribute values from the whole summation, we can say that the combination will not get maximum

score.

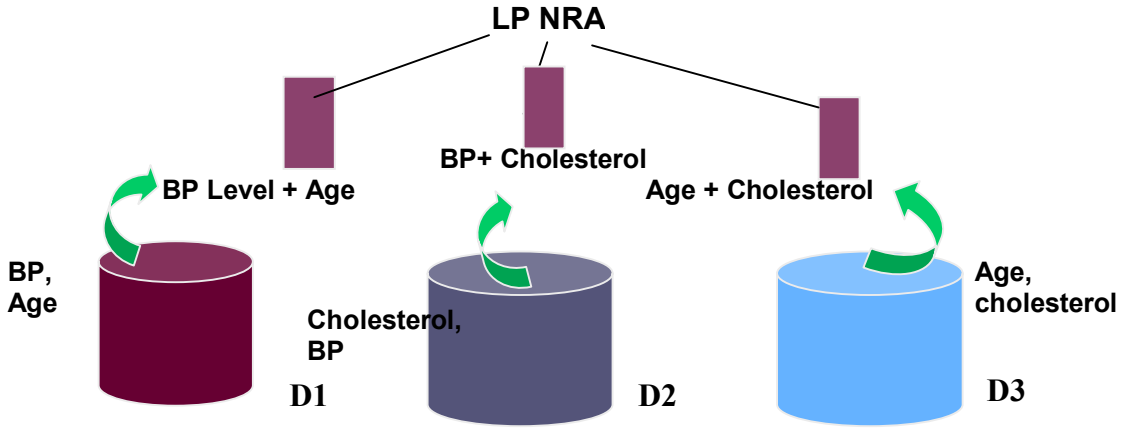

Fig 12 Functioning of Linear programming NRA with three databases having vertical overlapping and fragmentation.

In the above figure, we can see that we have BP and Age attributes overlapping in D1 and D2 while Age and Cholesterol attributes overlapping in D2 and D3 databases. In such cases, we can query all these databases according to the ranking that is already performed by the individual databases with respect to the scoring function provided by the query. This gives us the already sorted tuples faster than individual ones. Also since the combined NRA will not get the tuples according to the overlapped attributes but only get the overlapped attributes once, they will have less percentage of already ranked tuples.

One important thing to keep in mind here is that we need to have all the tuples in all databases even if the attributes are not the same. This is to ensure correct result is obtained by solving linear program. Linear program assumes that the input variables constraints will apply to the variable everywhere the situation occurs. For e.g. If we

have x1+x2>200, this condition should be satisfied no matter where it occurs. But in our case, one database may have x1+x2>200 while another may be having x1+x2>300 which will not be useful if we want to use linear programming. Hence we need all the tuples in each database.

The LP algorithm helps solve the problem of overlapping by determining the best score and the threshold value. Best Score value is the highest score any tuples can get. For this, the maximum score for all unseen attributes of that tuples needs to be determined. Since the attributes are overlapping, we use linear programming to get the maximum cost function. For e.g. If x1+x2+x3>200 and x1+x2+x4 > 500 are two constraints, we can use linear programming to find out the maximize cost function x1+x2+x3+x4. Hence we can find the best score any tuples can get.

Similarly threshold can be found out by supplying the constraints from each database having overlapping attributes and aggregating with non-overlapping attributes.

4.2.6.3 LPNRA Algorithm

Let Q = (*ScoreQ, k,* ∗) – Query

*topk*Buffer = *List* *topkmin* = 0

for all database d*i*($1 \leq i \leq n$) loop

Let (*tid,* $Score_Q$ *(tid)*) be the ith tuple from dj

// check viability of the tuple to be in *topk*-Buffer

if *ScoreQ*(*tid*) > *topkmin* then

if (|*topk*Buffer| = *k*) then

Remove min score tuple from *topk*Buffer

end if

Add (*tid, ScoreQ*(*tid*)) to *topk*-Buffer

topkmin = min score of *topk*Buffer

end if

//check for stopping condition

Compute *threshold* = LP ((*ScoreQd)* $\leq sd$)

where $1 \leq d \leq n$ *and sd is actual score*

$0 \leq Xj \leq 1$ $1 \leq j \leq m$ *(m attributes in cost function)*

if (|*topk*Buffer| = *k*) and (*unseenmax* $\leq$ *topkmin*) then

Return *topk*Buffer

end if

end for

CHAPTER 5

EXPERIMENTAL EVALUATION

5.1 Platforms

In our experimental evaluation, we present the methodologies for our experiment. We used Intel Pentium 2 Ghz processor with 1 MB memory with Windows XP SP2 o/s. We performed the experiment using Microsoft Visual Studio.Net with Windows form and C#.Net. Our Database constitutes of around 20000 records from Yahoo Autos. We generated synthetic data for increasing our attribute lists to 15. We normalized the data to have the upper bound 1 and lower bound as 0 for all the attributes.

5.1.1 Linear Programming

We used the API provided by extreme optimization for finding out the solution to the linear programming. It provides the complete platform for technical and statistical computing for .Net 2.0 platform. It has math, vector, matrix and statistics library in one package.

We have used the library provided for Optimization by using State of art algorithms for finding the minimum or maximum of a function in one or more variables.

This is under the package of "Extreme.Mathematics.Optimization.LinearProgramming". It provides a Linear Program class where one can add the cost function, constraints with

coefficients and variables and the inequalities. The solve () method will calculate the cost function according to our requirement of maximization or minimization. We tested the results with various other linear programs provided by educational and vendor sites.

This solution uses the simplex method to calculate the solution. The primal problem has an extreme point at the intersection of any two constraints, including the non-negativity constraints. Each extreme point is called a basic solution but only the points in the feasible region are called the basic feasible solutions. An *optimum* basic feasible solution for the primal problem maximizes the objective function P on the feasible set and that is what we get as the solution for our linear programming method.

5.2 Methodology

We conducted series of run varying the data base sizes and their horizontal and vertical fragmentation. In the first experiment, we have 10 data sources with the following schemas for total 10 dimensions.

D1 (A, B) Record size:

D2 (B, C, D)

D3 (C, D, E, F)

D4 (E, F)

D5 (G, H, I, J)

D6 (G, H)

D7 (I, J)

D8 (C, D, E, F, G, H)

D9 (I, J)

D10 (I, J)

We had input box for every database to vary their record sizes so as to vary the horizontal fragmentation.

For the second experiment, we took 5 data sources with 15 dimensions and with the schema as follows.

D1 (A, B)

D2 (B, C, D)

D3 (E, F)

D4 (G, H, I, J)

D5 (I, J, K, L, M, N, O)

The overlapping attributes were present in both the experiments. For each of the data sources, we took a series of experiments with varying database size and top k value. In Fig 11 shows the performance of both INRA and CNRA. As we can observe that the CNRA performs significantly less data access than INRA. Also the k value increase will steadily increase the access of database in LPNRA but it does not have a drastic impact on INRA as well. Similarly Fig 12 shows the comparison between CNRA, LPNRA and INRA as the k increase. This is based on only vertical fragmentation and overlapping to include LPNRA as well. We observed that the INRA improved with k value, as it already has most of the top candidates in the buffer while LPNRA is able to quickly find the top elements and so it doesn't have to scan deep enough in the list for first few tuples but as k increases, it will need to do more database access as the buffer doesn't contain much of data.

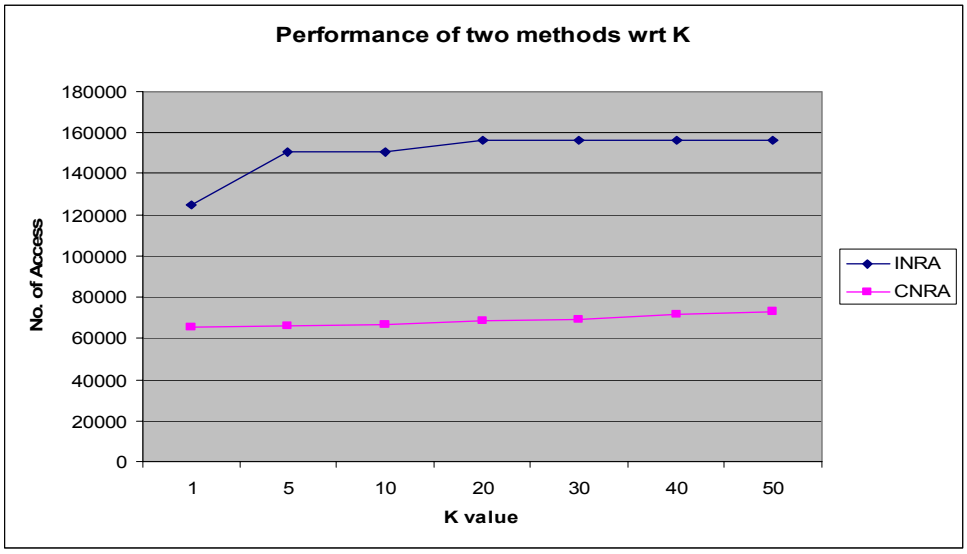

Fig 13 Comparison of INRA and CNRA based on varying k value

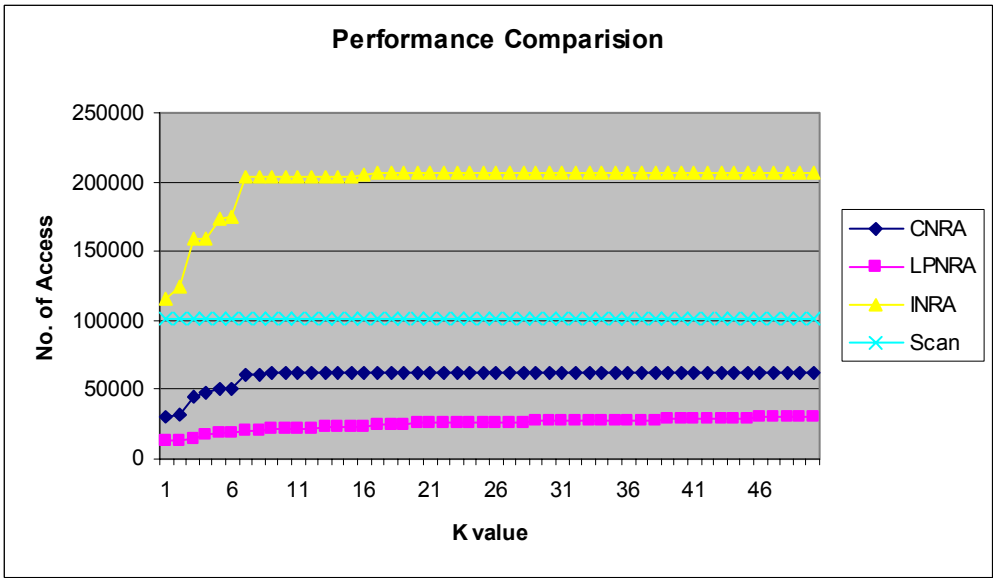

Fig 14 Comparison of CNRA, LPNRA and INRA with varying k value

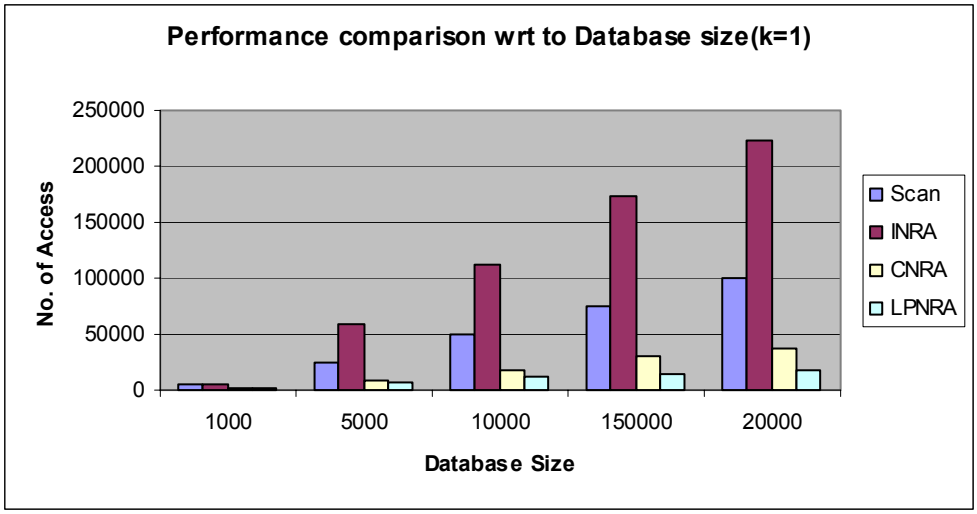

Fig 15 Comparison of Scan, INRA, CNRA and LPNRA with varying database size

The INRA database access increased drastically with the increase in the record size, while LPNRA was the least to increase even though they had the same database schema.

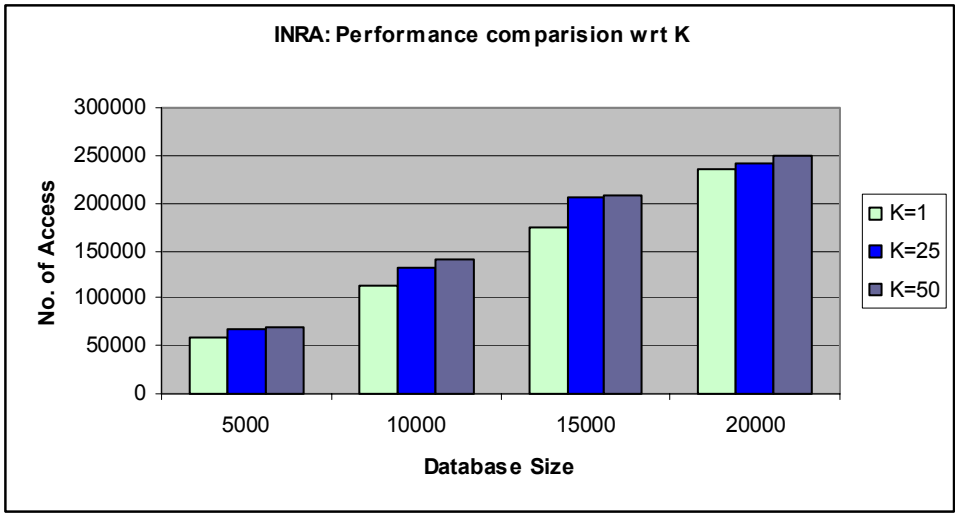

Fig 16 Comparison of performance by INRA with varying k and database size

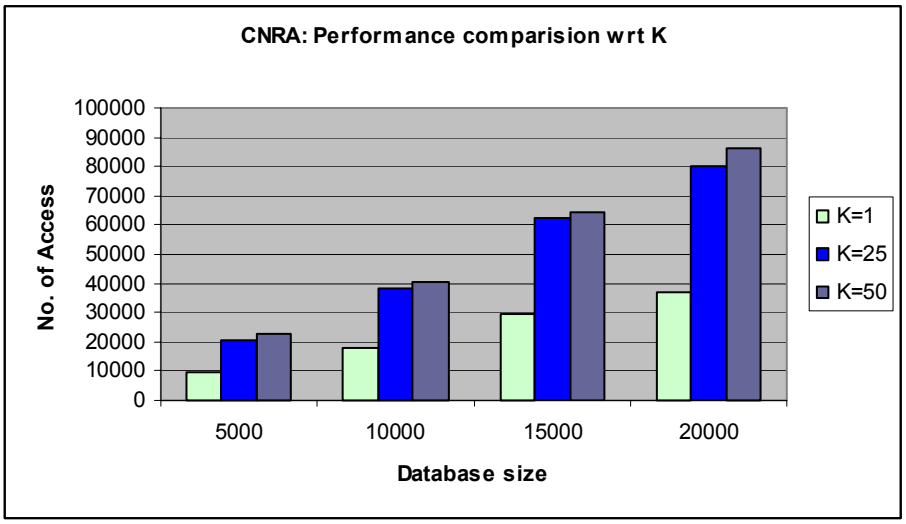

Fig 17 Comparison of performance by CNRA with varying k and database size

In the figure 14, comparison of Combined NRA with the database size and k size, we see that the performance decreases with database size quite consistently whatever the k size is.

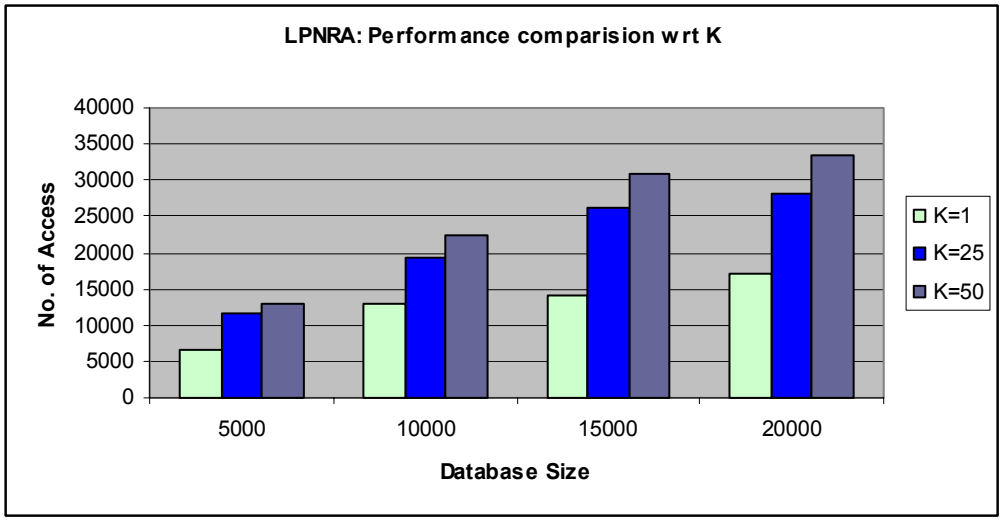

Fig 18 Comparison of performance by LPNRA with varying k and database size

CHAPTER 6

CONCLUSION

We have focused on a very novel but realistic problem in this research. Previous research work has not been done in the case of overlapping database and taking combined attributes as sorted lists. Even though the problem is unique and very interesting, it is quite complex due to its unpredictable nature of fragmentations and overlapping as well as the distributed environment where we need to keep track of schema matching problems. Most of the previous work focused on minimizing the query rounds using batch processing and it's variation on pruning the candidates. Also significant work has been focused towards the ranking aware query optimization and also moving of the computation to different nodes. They have considered overlapping in the sense of duplicate records but not duplicate records with partial combined attributes in multiple databases. Our algorithm has invariably less access cost than the general top k selection algorithms since we take advantage of already ranked tuples and overlapping attributes in the local databases.

For the future scope, we can use the batch processing instead of getting individual tuples from each database. In this way we can reduce the latency occurrence for querying each peer. Also one of the obvious drawbacks here is the increase in per-peer load. We can evaluate the cost for this drawback and see if this is worth pursuing

and if we can generalize and automate the system to capture the most efficient algorithms among the CNRA, INRA and LPNRA depending on the fragmentations of our distributed databases. We can have a schema controller which keeps the schemas of all the database and helps in schema matching. This will help automate the finding of combined attributes as well as the missing attributes.

Also we can take approximate answers instead of exact answers by pruning of candidates early in the stage using probabilistic methods or using the database statistics like histogram structures.

We can further refine our experiment results by adding the performance wrt dimension expansion and also the performance wrt to data source expansion.

REFERENCES

1) Gautam Das, Dimitrios Gunopulos, Nick Koudas: Answering Top-k Queries Using Views VLDB 2006

2) Nicolas Bruno, Luis Gravano und Am´elie Marian: Evaluating Top-k Queries over Web-Accessible Databases. In ICDE, 2002.

3) Hailing Yu, Hua-Gang Li, Ping Wu, Divyakant Agrawal, Amr El Abbadi: Efficient Processing of Distributed Top-k Queries *DEXA 2005*

4) Ulrich Guntzer, Wolf-Tilo Balke und Werner Kiesling : Optimizing Multi-Feature Queries for Image Databases, VLDB 2000

5) Ulrich G¨untzer, Wolf-Tilo Balke und Werner Kiesling: Towards Efficient MultiFeature Queries in Heterogenous Environments ITCC 2001

6) I.F. Illyas, W. G. Aref, A. k. Elmagarmid: Supporting top k join Queries in Relational databases, VLDB 2004

7) A. Marian, N. Bruno, L. Gravano: Evaluating Top-k Queries over Web-Accessible Databases. ACM TODS

8) Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, Jeffrey Scott Vitter : Supporting incremental join Queries on Ranked inputs VLDB 2001

9) Martin Theobald, Gerhard Weikum, Ralf Schenkel: Top K query evaluation with probabilistic guarantees, VLDB 2004

10) P. Cao, Z. Wang: Efficient Top k Query Calculation in distributed Networks. PODC 2004

11) Martin Theobald, Ralf Schenkel, Gerhard Weikum: Efficient and Self tuning Incremental Query Expansion for Top k Query Processing, SIGIR 2005

12) Chengkai Li, Kevin chen-Chuan Chang, lhab F. llyas: RankSQL: Query Algebra and Optimization for Relational Top-k Queries

13) Holger Bast Debapriyo Majumdar Ralf Schenkel Martin Theobald Gerhard Weikum: IO-Top-k: Index-access Optimized Top-k Query Processing, VLDB 2006, Seoul, Korea.

14) Sebastian Michel, Peter Triantafillou, Gerhard Weikum : KLEE: A Framework for Distributed Top-k Query Algorithms VLDB Conference, Trondheim, Norway, 2005

15) Nikos Mamoulis, Kit Hung Cheng, Man Lung Yiu, and David W. Cheung : Efficient Aggregation of Ranked Inputs, ICDE 2006

16) Zhigang Chen, Zhongding Huang, Bo Ling, Jiang Li: P2P-Join: A Keyword Based Join Operation in Relational Database Enabled Peer-to-Peer Systems, IEEE 2006

17) Martin Theobald Ralf Schenkel Gerhard Weikum : TopX – Efficient and Versatile Top-k Query Processing for Text, Structured, and Semistructured Data

18) Surajit Chaudhuri, Luis Gravano: Evaluating Top-*k* Selection Queries: Proceedings of VLDB 1999

19) Ronald Fagin, Amnon Lotem, Moni Naor : Optimal aggregation algorithms for middleware

21) D. ZeinalipourYazti, Z. Vagena, D. Gunopulos, V. Kalogeraki, V. Tsotras, M. Vlachos N. Koudas D. Srivastava : The Threshold Join Algorithm for Topk Queries in Distributed Sensor Networks

BIOGRAPHICAL INFORMATION

Amrita Tamrakar was born in Kathmandu, capital of Nepal. She completed her B.E from REC, Durgapur in India after winning a scholarship from Indian Embassy. After working in Telecom industry in Nepal for few years, she pursued her Masters Degree in Computer Science and Engineering to fulfill her constant yearning to learn more on the database field.