ISSN:2348-2079

Volume-6 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Parallel and Multiple E-Data Distributed Process with Progressive Duplicate Detection Model

[1]Mrs. K. E. Eswari, M.C.A.,M.Phil., M.E., Associate Professor/MCA
[2]Mr. S.PraveenKumar, III MCA
Department of MCA ,Nandha Engineering College(Autonomous),Erode-52
E-Mail ID: eswari.eswarimoorthy@nandhaengg.org, praveenkumar.sb.sc@gmail.com

***Abstract*- Duplicate detection is the process of identifying multiple representations of same real world entities. In present, duplicate detection methods need to process ever larger datasetsin ever shorter time: maintaining the quality of a dataset becomes increasingly difficult. This paper presents two novel, progressive duplicate detection algorithms that significantly increase the efficiency of finding duplicates if the execution time is limited: They maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches. Comprehensive experiments show that progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work. Data are among the most important assets of a company. But due to data changes and sloppy data entry, errors such as duplicate entries might occur, making data cleansing and in particular duplicate detection indispensable. Progressive duplicate detection identifies most duplicate pairs early in the detection process.**

***Index Term*-Duplicate detection, entity resolution, pay-as-you-go, progressiveness, data cleaning**

I.INTRODUCTION

Data mining, or knowledge discovery, is the computer-assisted process of digging through and analyzing enormous sets of data and then extracting the meaning of the data. Data mining tools predict behaviors and future trends, allowing businesses to make proactive, knowledge-driven decisions.Data mining tools can answer business questions that traditionally were too time consuming to resolve. They scour databases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations. Data mining derives its name from the similarities between searching for valuable information in a large database and mining a mountain for a vein of valuable ore. Both processes require either sifting through an immense amount of material, or intelligently probing it to find where the value resides.

- To identify the multiple representations of same real world entities
- To propose progressive duplicate detection algorithms that significantly increases the efficiency of finding duplicates if the execution time is limited.
- To maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches.
- To double the efficiency over time of traditional duplicate detection and significantly improve upon related work
- To use concurrent approach. i.e., all the records are taken and checked as a parallel processes.
- To reduce Execution time.

II. RELATED WORKS

Steven EuijongWhang et al [1] describe the Entity resolution (ER) is the problem of identifying which records in a database refer to the same entity. In practice, many applications need to resolve large data sets efficiently, but do not require the ER result to be exact. For example, people data from the Web may simply be too large to completely resolve with a reasonable amount of work. As another example, real-time applications may not be able to tolerate any ER processing that takes longer than a certain amount of time. This paper investigates how we can maximize the progress of ER with a limited amount of work using "hints," which give information on records that are likely to refer to the same real-world entity. A hint can be represented in various formats (e.g., a grouping of records based on their likelihood of matching), and ER can use this information as a guideline for which records to compare first. We introduce a family of using the hints to maximize the number of matching records identified using a limited amount of work. Using real data sets, we illustrate the

potential gains of our pay-as-you-go approach compared to running ER without using hints.

JayantMadhavan, [2] describe the World Wide Web is witnessing an increase in the amount of structured content – vast heterogeneous collections of structured data are on the rise due to the Deep Web, annotation schemes like Flickr, and sites like Google Base. While this phenomenon is creating an opportunity for structured data management, dealing with heterogeneity on the web-scale presents many new challenges. In this paper, we highlight these challenges in two scenarios – the Deep Web and Google Base. We contend that traditional data integration techniques are no longer valid in the face of such heterogeneity and scale. We propose a new data integration architecture, PAYGO, which is inspired by the concept of data spaces and emphasizes pay-as-you-go data management as means for achieving web-scale data integration. Chuan Xiao [3] describe the ssimilarity join is a useful primitive operation underlying many applications, such as near duplicate Web page detection, data integration, and pattern recognition.

Traditional similarity joins require a user to specify a similarity threshold. In this paper, we study a variant of the similarity join, termed top-k set similarity join. It returns the top-k pairs of records ranked by their similarities, thus eliminating the guess work users have to perform when the similarity threshold is unknown before hand. An algorithm, top k-join, is proposed to answer top-k similarity join efficiently. It is based on the prefix fliteringg principle and employs tight upper bounding of similarity values of unseen pairs. Experimental results demonstrate the efficiency of the proposed algorithm on large-scale real datasets.

Draisbach et al [4] describe the duplicate detection is the process of finding multiple records in a dataset that represent the same real-world entity. Due to the enormous costs of an exhaustive comparison, typical algorithms select only promising record pairs for comparison. Two competing approaches are blocking and windowing. Blocking methods partition records into disjoint subsets, while windowing methods, in particular the Sorted Neighborhood Method, slide a window over the sorted records and compare records only within the window. We present a new algorithm called Sorted Blocks in several variants, which generalizes both approaches. To evaluate Sorted Blocks, we have conducted extensive experiments with different datasets. These show that our new algorithm needs fewer comparisons to find the same number of duplicates.

Manolis Wallace et al [5] describe the fieldoftransitive relations focuses mainly on dense, Boolean, undirected relations. With the emergence of a new area of intelligent retrieval, where sparse transitive fuzzy ordering relations are utilized, existing theory and methodologies need to be extended, as to cover the new needs. This paper discusses the incremental update of such fuzzy binary relations, while focusing on both storage and computational complexity issues.Moreover, it proposes a novel transitive closure algorithm that has a remarkably low computational complexity (below O(n2)) for the average sparse relation; such are the relations encountered in intelligent retrieval.

Ahmed K. Elmagarmid [6] describe the data cleaning [2], or data scrubbing [3], refers to the process of resolving such identification problems in the data. We distinguish between two types of data heterogeneity: structural and lexical. Structural heterogeneity occurs when the fields of the tuples in the database are structured differently in different databases. For example, in one database, the customer address might be recorded in one field named, say, address, while, in another database, the same information might be stored in multiple fields such as street, city, state, and zipcode. Lexical heterogeneity occurs when the tuples have identically structured fields across databases, but the data use different representations to refer to the same real-world object. In this paper, we focus on the problem of lexical heterogeneity and survey various techniques which have been developed for addressing this problem.

We focus on the case where the input is a set of structured and properly segmented records, i.e., we focus mainly on cases of database records. Hence, we do not cover solutions for various other problems, such as that of mirror detection, in which the goal is to detect similar or identical Web pages. Also, we do not cover solutions for problems such as anaphora resolution [6] in which the problem is to locate different mentions of the same entity in free text.

We should note that the algorithms developed for mirror detection or for anaphora resolution are often applicable for the task of duplicate detection. Techniques for mirror detection have been used for detection of duplicate database records and techniques for anaphora resolution are commonly used as an integral part of de duplication in relations that are extracted from free text using information extraction systems.

III.SYSTEM METHODOLOGY

A. INPUT PARAMETERS (D, K, W, I, N)

In this module, input for the Algorithm PSNM is selected. The algorithm takes five input parameters: D is a reference to the data, which has not been loaded from disk yet.The sorting key K defines the attribute or attribute combination that should be used in the sorting step. W specifies the maximum window size, which corresponds to the window size of the traditional sorted neighborhood method. When using early termination, this parameter can be set to an optimistically high default value. Parameter I defines the enlargement interval for the progressive iterations. N is the number of records.

B.INPUT PARAMETERS (D, K, R, S, N)

In this module, input for the Algorithm PB is selected. The algorithm takes five input parameters: D is a reference to the data, which has not been loaded from disk yet. The sorting key K defines the attribute or attribute combination that should be used in the sorting step. R specifies the maximum block range, S Block Size and N Total No. of Records. When using early termination, this parameter can be set to an optimistically high default value.

C. PROGRESSIVE SORTED NEIGHBORHOOD METHOD ALGORITHM

The PSNM algorithm calculates an appropriate partition sizei.e., the maximum number of records that fit in memory, using the pessimistic sampling function calcPartitionSize(D) in Line 2: If the data is read from a database, the function can calculate the size of a record from the data types and match this to the available main memory. Otherwise, it takes a sample of records and estimates the size of a record with the largest values for each field.Require: dataset reference D, sorting key K, window sizeW, enlargement interval size I, number of records N

```
procedure PSNM(D, K, W, I, N)
pSizecalcPartitionSize(D)
pNum  N←Size /W þ 1Þ de
array order size N as Integer
array recs size pSize as Record
ordersortProgressive(D, K, I, pSize, pNum)
forcurrentI 2 to W I de do
forcurrentP 1 to pNum do
recsloadPartition (D, currentP)
fordist 2 range(currentI, I, W) do
fori 0 to recs jj←dist do
pair  recs½i← recs[i] ←dist+hi
if compare(pair) then
emit(pair)
lookAhead(pair)
```

In Line 3, the algorithm calculates the number of necessary partitions pNum, while considering a partition overlap of W - 1 records to slide the window across their boundaries. Line 4 defines the order-array, which stores the order of records with regard to the given key K. By storing only record IDs in this array, we assume that it can be kept in memory. To hold the actual records of a current partition, PSNM declares the recs-array in Line 5.

In Line 6, PSNM sorts the dataset D by key K. The sorting is done using progressive sorting algorithm. Afterwards, PSNM linearly increases the window size from 2 to the maximum window size W in steps of I (Line 7). In this way, promising close neighbors are selected first and less promising far-away neighbors later on. For each of these progressive iterations, PSNM reads the entire dataset once. Since the load process is done partition-wise, PSNM sequentially iterates (Line 8) and loads (Line 9) all partitions. Require: dataset reference D, key attribute K, maximum block range R, block size S and record number N

```
procedure PB(D, K, R, S, N)
pSizecalcPartitionSize(D)
bPerPpSize S bc
bNum  N S de
pNumbNumbPerP de
array order size N as Integer
array blocks size bPerP as Integer Record← hi
priority queue bPairs as Integer IntegerInteger hi
bPairs  1 1 hi ... bNumbNum hi fg
ordersortProgressive(D, K, S, bPerP, bPairs)
fori 0 to pNum← 1 do
pBPs get(bPairs, i←bPerP,(i þ 1) ←bPerP)
blocksloadBlocks(pBPs, S, order)
compare(blocks, pBPs, order)
whilebPairs is not empty do
pBPsfg
bestBPstakeBest( bPerP 4 bc, bPairs, R)
forbestBP 2 bestBPs do
ifbestBP[1] ←bestBP[0] R then
pBPspBPs [ extend(bestBP)
blocksloadBlocks(pBPs, S, order)
compare(blocks, pBPs, order)
bPairsbPairs [ pBPs
procedure compare(blocks, pBPs, order)
forpBP 2 pBPs do
dPairscNum hi comp(pBP, blocks, order)
emit(dPairs)
pBP[2]  dPairsjj / cNum
```

To process a loaded partition, PSNM first iterates overall record rank-distances dist that are within the current window interval current I. For I= 1 this is only one distance, namely the record rank-distance of the current main-iteration. In Line 11, PSNM then iterates all records in the current partition to compare them to their dist-neighbor. The comparison is executed using the compare (pair)function in Line13**.** If this function returns "true", a duplicate has been found and can be emitted.

IV. BLOCKING TECHNIQUES

A block pair consisting of two small blocks defines only few comparisons. Using such small blocks, the PB algorithm carefully selects the most promising comparisons and avoids many less promising comparisons from a wider neighborhood. Block pairs based on small blocks cannot characterize the duplicate density in their neighborhood well, because they represent a too small sample. A block pair consisting of large blocks, in contrast, may define too many, less promising comparisons, but produces better samples for the extension step. The block size parameter S, therefore, trades off the execution of non-promising comparisons and the extension quality.

MagpieSort. To estimate the records' similarities, the PB algorithm uses an order of records. As in the PSNM algorithm, this order can be calculated using the progressive MagpieSort algorithm. Since each iteration of this algorithm delivers a perfectly sorted subset of records, the PB algorithm can directly use this to execute the initial comparisons.

A. Attribute Concurrent PSNM

The basic idea of AC-PSNM is to weight and re-weight all given keys at runtime and to dynamically switch between the keys based on intermediate results. Thereto, the algorithm pre-calculates the sorting for each key attribute. The pre-calculation also executes the first progressive iteration for every key to count the number of results. Afterwards, the algorithm ranks the different keys by their result counts. The best key is then selected to process its next iteration. The number of results of this iteration can change the ranking of the current key so that another key might be chosen to execute its next iteration.

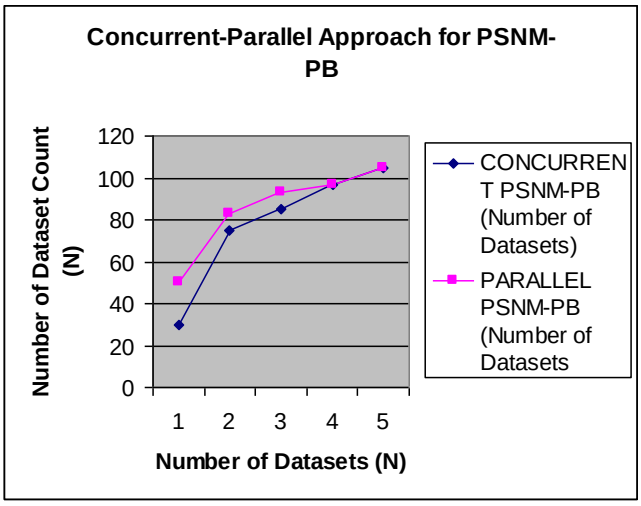

Fig1.Concurrent-Parallel Approach for PSNM-PB

S.NO	DATASETS	CONCURRENT PSNM-PB (Number of Datasets)	PARALLEL PSNM-PB (Number of Datasets
1	100	30	50
2	200	75	83
3	300	85	93
4	400	97	97
5	500	105	105

Table 1. Concurrent-Parallel Approach for PSNM-PB

V. CONCLUSION

Through this project, the efficiency of duplication detection is increased better than existing system. This project introduced the progressive sorted neighborhood method and progressive blocking. Both algorithms increase the efficiency of duplicate detection for situations with limited execution time they dynamically change the ranking of comparison candidates based on intermediate results to execute promising comparisons first and less promising comparisons later. The project proposed a novel quality measure for progressiveness that integrates seamlessly with existing measures. It uses multiple sort keys concurrently to interleave their progressive iterations. By analyzing intermediate results, both approaches dynamically rank the different sort keys at runtime, drastically easing the key selection problem. It is believed that almost all the system objectives that have been planned at the commencements of the software development have been net with and the implementation process of the project is completed. A trial run of the system has been made and is giving good results the procedures for processing is simple and regular order. The process of preparing plans been missed out which might be considered for further modification of the application.

REFERENCES

[1] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," IEEE Trans. Knowl. Data Eng., vol. 19, no. 1, pp. 1–16, Jan. 2007.

[2] F. Naumann and M. Herschel, An Introduction to Duplicate Detection. San Rafael, CA, USA:Morgan& Claypool, 2010.

[3] H. B. Newcombe and J. M. Kennedy, "Record linkage: Making maximum use of the discriminating power of identifying information," Commun. ACM, vol. 5, no. 11, pp. 563–566, 1962.

[4] M. A. Hernandez and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," Data Mining Knowl. Discovery, vol. 2, no. 1, pp. 9–37, 1998.

[5] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller, "Framework for evaluating clustering algorithms in duplicate detection," Proc. Very Large Databases Endowment, vol. 2, pp. 1282–1293, 2009.

[6] O. Hassanzadeh and R. J. Miller, "Creating probabilistic databases from duplicated data," VLDB J., vol. 18, no. 5, pp. 1141–1166, 2009.

[7]U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg, "Adaptive windows for duplicate detection," in Proc. IEEE 28thInt. Conf. Data Eng., 2012, pp. 1073–1083.