**PARALLEL AND MULTIPLE E-DATA DISTRIBUTED PROCESS WITH PROGRESSIVE DUPLICATE DETECTION MODEL**

V.Yasvanthkumaar,

M.Tech Information Technology,

Department of Information Technology

K.S.R. College of Engineering,

Tiruchengode, Tamilnadu.

Yasvanth123456@gmail.com

## 

***Abstract***

In present, duplicate detection methods need to process ever larger datasets in ever shorter time: It is difficult to maintain the dataset. This project presents progressive duplicate detection algorithm that gradually increase the efficiency of finding duplicates if the execution time is limited: They maximize the gain of the overall process within the available time by reporting most results. These experiments show that progressive algorithms can double the efficiency over time of traditional duplicate detection and improve the work. Progressive duplicate detection identifies most duplicate pairs in the detection process. Instead of reducing the overall time needed to finish the entire process, this approaches tries to reduce the average time.

**I. INTRODUCTION**

Data mining, is the process of extraction of data from huge database. Data mining tools can answer business questions that were too time consuming. They analyze databases for hidden patterns that experts may not predict directly. Data mining derives its name from the similarities between searching for valuable information from dataset.

* 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 available time by reporting most results earlier than traditional approaches.
* To double the efficiency over the 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.
* To make use of Resource consumption which is same as existing system but the data is kept in multiple resource memories.

**II.RELATED WORKS**

Steven Euijong Whang et al [1] describe the Entity resolution (ER) is the problems of identifying 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. 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 the progress of ER with a limited amount of work which gives information on records that are likely to refer to the same real-world entity. We introduce a family of using the hints to maximize the number of matching records identiﬁed by the 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.

Jayant Madhavan, [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 .This phenomenon is used for structured data management, dealing with heterogeneity on the web-scale challenges. We contend that traditional data integration techniques are not valid in heterogeneity and scale. A new data integration architecture, PAYGO, is proposed, which is inspired by the data spaces and make emphatics pay-as-you-go data management which means for achieving web data integration.

Chuan Xiao [3] describe the similarity join is a useful primitive operation underlying many applications, such as duplicate detection in web page, data integration, and pattern recognition. In traditional similarity joins, it requires a user to specify a similarity threshold. In this paper, we study a variant of the similarity join, termed as top-k set similarity join. It returns the top-k pairs of records classified by their similarities, thus eliminating the guess work users have to perform when the similarity threshold is unknown earlier. An algorithm, top k-join, is proposed to answer top-k similarity join efficiently. It is based on the preﬁx flitering principle and employs tight upper bounding of similarity values of unseen pairs.

Draisbach et al [4] describe the duplicate detection is the process of finding several records in a dataset that represent the same real-world entity. Two competing approaches are blocking and windowing. Blocking methods can partition records into multiple subsets, while windowing methods, in particular the Sorted Neighborhood Method uses slide a window over the sorted records and compare those records only within the window. These show that our new algorithm needs fewer comparisons to find the same number of duplicates.

Manolis Wallace et al [5] describe the ﬁeld of transitive relations focuses mainly on dense, Boolean, undirected relations. With the occurrence of a new area of intelligent retrieval, where they utilize sparse transitive fuzzy ordering relations, existing theory and methodologies need to be extended. This paper discusses the incremental update of fuzzy binary relations, while focus on both storage and computational complexity problem. Moreover, it proposes a novel transitive closure algorithm that has a remarkably low computational complexity for the average sparse relation; such are the relations encountered in intelligent retrieval.

**III. METHODOLOGY**

**INPUT PARAMETERS (D, K, W, I, M)**

In this module, input for the Algorithm PSNM is selected. The algorithm takes five input parameters: D is a reference to the data. The sorting key K defines the attribute or attribute combination that should be used in the sorting step and W specifies the maximum window size. When using early termination, this parameter can be set to an high default value. Parameter I defines the enlargement interval for the progressive iterations. M is the number of records.

**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 its sequence 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 completion, this parameter can be set to an optimistically high default value.

**PROGRESSIVE SORTED NEIGHBORHOOD METHOD ALGORITHM**

In this phase, algorithm calculates an **appropriate partition size** pSize, i.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 will evaluate the size of a record and fix 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: reference to the data D, sorting key K, window size

W, enlargement interval size I, number of records M

1: **procedure** PSNM (D, K, W, I, M)

2: pSize 🡨 calcPartitionSize(D)

3: pNum 🡨[ M/(pSize - W + 1) ]

4: **array** order **size** M **as** Integer

5: **array** recs **size** pSize **as** Record

6: order 🡨 sortProgressive(D, K, I, pSize, pNum)

7: **for** currentI 🡨 2 **to** [W/ I] **do**

8: **for** currentP 🡨 1 **to** pNum **do**

9: recs 🡨 loadPartition (D, currentP)

10: **for** dist ∈ range(currentI, I, W) **do**

11: **for** i 🡨 0 **to** |recs| - dist **do**

12: pair 🡨〈 recs[i], recs[i + dist]〉

13: **if** compare(pair) **then**

14: emit(pair)

15: lookAhead(pair)

In Line 3, the algorithm calculates pNum, the number of necessary partitions, and consider a partition overlap of W - 1 record to slide the window. Line 4 defines the order- array, that stores order of records respect 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 array in Line 5.

PSNM sorts the dataset D by key K in Line 6,. The sorting is done using progressive sorting algorithm. Then, PSNM linearly increases the window size W gradually in steps of I (Line 7). In this way, assuring close neighbors are selected first and less promising far-away neighbors later on. For each progressive iteration, PSNM reads the entire dataset sequentially. Since the load process is done partition-wise, PSNM [continuously](http://www.wordhippo.com/what-is/another-word-for/continuously.html) iterates (Line 8) and loads (Line 9) all partitions. Require: dataset reference D, key attribute K, maximum block range B, block size S and record number N

**PROGRESSIVE BLOCKING ALGORITHM**

**Require:** dataset reference D,key attribute K,maximum block range B,block size S and record number N

1: **procedure** PB(D, K, B, S, N)

2: pSize 🡨 CalcPartitionSize (D)

3: bPerP 🡨 [ pSize / S ]

4: bNum 🡨[ N / S ]

5: pNum 🡨[ bNum/ bPerP ]

6: **array** order **size** N **as** Integer

7: **array** block **size** bPerP **as** 〈Integer, Record[ ]〉

8: **priority queue** bPairs as 〈Integer, Integer, Integer 〉

9: bPairs 🡨{〈 1, 1,..〉 ,... ,〈 bNum, bNum,-〉}

10: order 🡨 sortProgressive(D, K, S, bPerP, bPairs)

11: **for** i 🡨 0 **to** pNum - 1 **do**

12: pBPs 🡨 get(bPairs, i . bPerP,(i + 1) . bPerP)

13: blocks 🡨 loadBlocks(pBPs, S, order)

14: compare(blocks, pBPs, order)

15: **while** bPairs is not empty **do**

16: pBPs 🡨 { }

17: bestBPs 🡨 takeBest( ⎣ bPerP / 4 ⎦, bPairs, B)

18: **for** bestBP ∈ bestBPs **do**

19: **if** bestBP[1] - bestBP[0] < R **then**

20: pBPs 🡨 pBPs ∪ extend(bestBP)

21: blocks 🡨 loadBlocks(pBPs, S, order)

22: compare(blocks, pBPs, order)

23: bPairs 🡨 bPairs ∪ pBPs

24: **procedure** compare(block, pBPs, order)

25: **for** pBP ∈ pBPs **do**

26: 〈 dPairs, cNum 〉 🡨 comp(pBP, block, order)

27: emit(dPairs)

28: pBP[2] 🡨| dPairs | / cNum

To process a loaded partition, PSNM first iterates overall record rank-distances 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 exact 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**

Block size. 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 less promising comparisons from a wider neighborhood. The block pairs based on small blocks could not 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.

**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.

**V.CONCLUSION AND FUTURE ENHANCEMENT**

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. Hence algorithms increase the efficiency of duplicate detection for situations with limited execution time. 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. A trial run of the system has been made and is giving good results and 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.

In future work, to combine our progressive approaches with scalable approaches for duplicate detections to deliver the results even faster is analyzed. In particular, a two phase parallel SNM is introduced, which executes a traditional SNM on balanced, overlapping partitions.

**VII REFERENCES**

1. S. E.Whang, D. Marmaros, and H. Garcia-Molina, “Pay-as-you-go entity resolution,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 5,pp. 1111–1124, May 2012.
2. 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.
3. F. Naumann and M. Herschel, An Introduction to Duplicate Detection. San Rafael, CA, USA:Morgan & Claypool, 2010.
4. C.Xiao, W.Wang, X.Lin, and H.Shang, “Top-k set similarity joins,” in Proc. IEEE Int. Conf. Data Eng., pp. 916-927, 2009.
5. M. A. Hern andez 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, 2008.
6. X. Dong, A. Halevy, and J.Madhavan, “Reference reconciliation in complex information spaces,” in Proc. Int. Conf. Manage. Data, 2005, pp. 85–96.
7. 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.
8. O. Hassanzadeh and R. J. Miller, “Creating probabilistic databases from duplicated data,” VLDB J., vol. 18, no. 5, pp. 1141–1166, 2009.
9. 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.
10. S. Yan, D. Lee,M.-Y. Kan, and L. C. Giles, “Adaptive sorted neigh-borhood methods for efﬁcient record linkage,” in Proc. 7th ACM/IEEE Joint Int. Conf. Digit. Libraries, 2007, pp. 185–194.J.Madhavan,S.R.Jeffery.