Multiple Approach to Detect Duplicates in CD-Dataset

Bondugulapati Keerthana[1], Dr. K. Ramakrishna[2]

[1]Department of CSE, CMR College of Engineering & Technology, Hyderabad, Telangana, India.
Email: keerthanab1992@gmail.com
[2]Department of CSE, CMR College of Engineering & Technology, Hyderabad, Telangana, India.
Email: krkrishna.cse@gmail.com

Abstract**−***In Data mining, we are getting data from the cloud databases and the number of large size datasets will be increased in the cloud. Hence, the user must clean the dataset before using it. During the process of data cleaning, duplicate detection is one phase. Now-a-days, the user wants to process the larger datasets in less time which is not possible in the existing system. There are a number of methods to detect duplicates in the datasets traditionally, but those are not time efficient and users cannot get accurate data results. In the existing system, we have used two methods, namely, 1) Progressive Sorted Neighborhood Method (PSNM), 2) Progressive Blocking (PB) Method. These two methods provide good quality in duplicate detection, but those methods are not time efficient. To overcome this disadvantage, in this paper we propose a time efficient Parallel Processing Method. This method is extended by the traditional Progressive Sorted Neighborhood Method only. This parallel processing method, detects the duplicate data faster than the existing methods. In our experiments, we can observe the processing time of the parallel method and normal me*

Keywords—Data Mining, Dataset, Duplicate Detection, Parallel Processing, Sorting Key.

I. INTRODUCTION

Databases play a major role in the latest IT based financial system. Many industries depend upon the accuracy of databases to carry out different operations. For this reason, the standard of information saved within the databases, can have huge amount price implications to a procedure which depends on expertise to operate and conduct trade. Data processing must be done whenever the duplicates need to be found from the dataset. Within the field of engineering, information mining takes its concepts from information Discovery(KDD). In the majority of the domains, duplication is changing into a significant threat. As a result of this duplication, the information received is more and therefore memory limitation becomes demanding. Therefore, the admin finds it troublesome to manage the information sets. The people keep their portfolio dynamically despite retailers provides several product catalogs. In an error-free process with perfectly easy information, the development of a comprehensive view of the data contains linking --in relational terms, joining-- 2 or more tables on their key fields. Unluckily, data usually deficit a specified, international identifier that will allow such an operation. Moreover, the data are neither carefully managed for quality nor defined in a uniform means across distinct data sources. Therefore, knowledge quality is a rule understood by a way of many causes, together with knowledge entry error (e.g., studet in the place of student), missing integrity constraints (e.g., allowing entries comparable to Employee-Age=567), and a pair of conventions for recording expertise to make things worse in independently managed databases, but the structure, semantics as well as underlying expectations about the information may just differ as good. Revolutionary duplicate detection recognizes most replica pairs early in the detection approach. Instead of reducing the overall time needed to terminate the complete process, revolutionary procedures may attempt to lessen the normal time after which a duplicate is determined. Duplicate detection is the method of settling on more than a few representations which are equal to real-world purpose in a knowledge source. The quality of replica detection, i.e., its effectiveness, scalability cannot be unobserved as that of the gigantic measurement of the database. The duplicate identification drawback has two features: First, more than one representation is no longer equal, but combine differences, equivalent to misspellings, converted locations, or lost values. This makes it difficult to realize these duplicates. Second, duplication detection is a very costly operation, because it requires the comparison of each feasible pair of duplicates using the traditional complex similarity calculate. Progressive methods make this exchange-off extra invaluable, as they deliver the whole outcome in shorter quantities of time. The Revolutionary Sorted Nearby procedure takes smooth dataset and find some replica files and Progressive Blocking take dirty datasets and realize significant duplicate files in the databases. And finally, in this paper we propose Parallel Processing method and our work extends by these sorting methods.

II. RELATED WORK

Dong et al. performed reproduction detection within the PIM area by a way of using relationships to propagate similarities from one duplicate classification to yet another. The important focus of their process is to develop effectiveness with the aid of using relationships. In contrast, we are aware of increasing effectively via using relationships. Before describing our method in detail we provide some definitions and show an illustration of our technique. Mostly, in the real world, entities have two or more extra depictions in databases. Duplicate records do not share a fashioned key and/or they incorporate blunders that make a duplicate matching a complex challenge. Error is offered as the result of transcription error, incomplete expertise, lack of usual formats, or any blend of those reasons. In this paper, we provide a thorough evaluation of the literature on duplicate report detection. We cover similarity metrics which can be used most of the time to notice similar area entries, and we proposed an extensive set of duplicate detection algorithms that can notice reproduction records in a database approximately. We also cover a couple of tactics for improving the organization and scalability of inexact duplicate detection algorithms. We conclude with the protection of existing tools and with a quick dialogue of the significant open problems in the discipline(S. Ramya and Palaninehru, 2015). The problem that we learnt has been identified for more than five years because of the file linkage or the file matching problem in the records community. The purpose of document matching is to identify the records in the equal or unique databases that confer with the same real-world entity, even if the files are identical.

The Pay-as-you-go method explores how we can maximize the development of ER with a restricted amount of effort making use of "hints", (S. E. Whang, D. Marmaros, and H. Garcia-Molina,2012) which presents expertise on documents which are prone to point out to the same object. A hint can be represented in one of a kind of designs (e.g., a clustering of records based on their possibility of matching), and ER can use this information as a guiding principle for which files to be evaluated first. The Pay-as-you-go technique to entity resolution, is the place where we acquire fractional results regularly" as a way to get the minimum outcome faster. An ER approach is very luxurious due to very large data sets and compute-intensive report comparisons.

2.1. Adaptive Approaches

Earlier work on replica detection focus on decreasing the overall runtime. Thereby, one of the important proposed algorithms is already in a position of assessing the high-quality of assessment candidates. The algorithms use this information to prefer the assessment candidates cautiously. For the equal rationale, other systems utilize adaptive windowing methods, which dynamically modify the window measurement relying on the quantity of not too long ago located duplicates. These adaptive abilities dynamically secure the efficiency of reproduction detection, but run for unique durations of time and may not maximize the affectivity for any given time slot.

2.1.1. The Drawbacks of Traditional Methods

1) These adaptive approaches dynamically give a boost to the organization of duplicate detection, but unlike our revolutionary procedures, they have to run for unique durations of time and can't limit the affectivity for any given time slot.
2) Wants to method giant dataset in less time
3) Quality of dataset is very complex

III. FRAMEWORK

The main aim of this paper is to detect the duplicate data in the different types of large and small datasets parallelly. In this paper, we are detecting duplicates on CD dataset. To detect duplicate data in the dataset, we follow three main steps,

- Pair selection
- Pair wise comparison
- Clustering

3.1. System Architecture

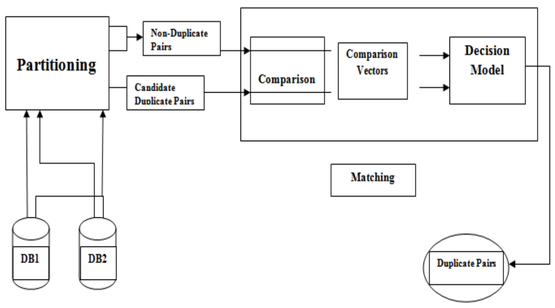


Fig.1: System Architecture

In the above architecture, we take some datasets and in the first step, we are partitioning our complete dataset. Partitioning is nothing but if we give a partition size=30 then it means that, we are keeping 30 records in every partition. After partitioning the dataset, we can perform the sorting algorithm on the dataset. In that sorting, it will compare the duplicates as a pair-wise comparison. After comparison, it will display the duplicate pairs to us.

3.2. Dataset Overview

In this paper, we are detecting the duplicates on CD dataset. It contains 9763 records and these records are related to the music and audio CDs. This dataset contains some of the attributes such as ID, artist, category, genre, cdextra, and year. From these attributes we can get some attributes as sorting keys by using the attribute concurrency method. For example, if we select “artist” as a sorting key then, the processing is done based on the artist related data only and after completion of processing it display the duplicate text of the artist attribute from dataset.

3.2.1. Sorting Key

1) Need of Sorting Key

Importance of this sorting key is that generally large dataset contains lakhs and thousands of records. Every time reading the complete dataset and detect all the duplicates in the dataset is not possible. Sometimes the user needs to detect the duplicate data and detect the duplicate count only on particular data. In this type of situations, we need a sorting key. Without sorting key it is difficult to sort the data from dataset.

To sort the dataset, we are using magpie sorting. In this sorting we are selecting one sorting key. To select the best key for sorting we are using the attribute concurrency method.

2) Sorting Key Selection

The best key for locating the duplicate is very hard to identify. Selecting good keys can increase the progressiveness. Here all the records are taken and checked as a parallel process so as to reduce the average execution time. The records are kept in multiple resources while splitting. The intermediate duplication results are intimated immediately when found in any resources and come back to most of the applications. Therefore the time consumption is decreased. Resource consumption is same as that of the existing system but the information is kept in multiple resource memories.

3.3. Algorithms

3.3.1. Progressive Sorted Neighborhood Method(PSNM)

```
1: procedure PSNM(D, K, W, I, N)
2:   pSize ← calcPartitionSize(D)
3:   pNum ← [N / (pSize – W + 1)]
4:   array order size N as Integer
5:   array recs size pSize as Record
6:   order ← sortProgessive(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)
```

3.3.2. Progressive Blocking (PB) Algorithm

```
1: procedure PB(D, K, R, 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 blocks 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).bPairs)
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, R)
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(blocks, pBPs, order)
25: **for** pBP ∈ pBPs **do**
26: <dPairs, cNum> ← comp(pBP, blocks, order)
27: emit(dPairs)
28: pBP[2] ← |dPairs| / cNum
```

3.4. Parallel Processing Method

Parallel processing means we execute the number of processes at a time that means parallelly. This is caused by using some of the concurrency methods. In this method first we are partition the dataset completely. These concurrency methods are used to execute all the partitions of the dataset at a time so as to reduce the execution time of the process. This proposed method selects the sorting key from the dataset by using the attribute concurrency method. It also takes the window/block size to partition the complete dataset. Basically, our proposed system is extended by the traditional Progressive Sorted Neighborhood Method (PSNM) and Progressive Block (PB). For that reason we have to give the window size as partition size. Based on these sorting key and window size, the parallel processing method executes all the partitions of the dataset and it also displays the parallel processing time of the proposed method.

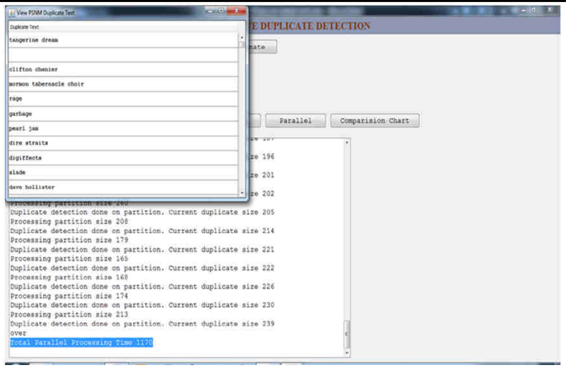
IV. EXPERIMENTAL RESULTS

In our experiments, we are going to detect duplicates on the CD-Dataset by using the Parallel Processing method. The first step in our experiment is to upload the CD-dataset into the system. After uploading the dataset, we must select the sorting key and the window/block size. This window/block size is used to partition the complete dataset and it is calculated by using this formula:

Partitions of Complete Dataset=Dataset Size / (Window/block size)

By using this formula, the size of each partition will be displayed and also it's duplicate size can be viewed in the system. And finally, the processing time of the algorithms is also displayed.

Here, we perform the traditional PSNM algorithm as well as traditional PB algorithm to verify the processing time of our proposed Parallel Processing method.



```
tangerine dream

clifton chenier
mormon tabernacle choir
rage
garbage
pearl jam
dire straits
digiffects
slade
dave hollister
```

```
ze 196
ze 201
ze 202
Processing partition size 260
Duplicate detection done on partition. Current duplicate size 205
Processing partition size 208
Duplicate detection done on partition. Current duplicate size 214
Processing partition size 179
Duplicate detection done on partition. Current duplicate size 221
Processing partition size 165
Duplicate detection done on partition. Current duplicate size 222
Processing partition size 168
Duplicate detection done on partition. Current duplicate size 226
Processing partition size 174
Duplicate detection done on partition. Current duplicate size 230
Processing partition size 213
Duplicate detection done on partition. Current duplicate size 239
over
Total Parallel Processing Time 1170
```

Fig. 2: Processing time of parallel processing method

The above screen shows the processing time of the Parallel Processing method.

The below screen shows the comparison chart for the normal processing time and parallel processing time:

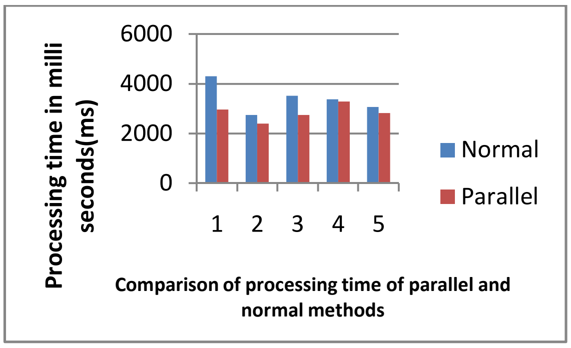


Fig. 3: Comparison chart for normal and parallel processing time

From our experiments, we observe that our proposed Parallel Processing method is a time efficient method to detect duplicates.

V. CONCLUSION

Finally, we conclude that in this paper we proposed a time efficient and improved Parallel Processing method. The proposed method is inspired by the traditional PSNM and PB algorithms. In our proposed method we get the duplicate detection time, duplicate count and duplicate text. In this experiment we used CD-Dataset and from this dataset we detect the duplicate count and duplicate text within milliseconds of time. Eventually, we proved that our proposed method is time efficient than the traditional algorithms.

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] 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.
[3] M. Wallace and S. Kollias, “Computationally efficient incremental transitive closure of sparse fuzzy binary relations,” in Proc. IEEE Int. Conf. Fuzzy Syst., 2004, pp. 1561–1565.
[4] P. Christen, “A survey of indexing techniques for scalable record linkage and deduplication,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 9, pp. 1537–1555, Sep. 2012.
[5] B. Kille, F. Hopfgartner, T. Brodt, and T. Heintz, “The Plista dataset,” in Proc. Int. Workshop Challenge News Recommender Syst., 2013, pp. 16–23.
[6] 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.
[7] S. Ramya and PalaninehruA, “Study of Progressive Techniques for Efficient Duplicate Detection”, 2015.
[8] R. Ramesh Kannan, D. R. Abarna, G. Aswini, P. Hemavathy, “Effective Progressive Algorithm for Duplicate Detection on Large Dataset”, 2016.

[9] B. Kille, F. Hopfgartner, T. Brodt, and T. Heintz, “The Plista dataset”, in Proc. Int. Workshop Challenge News Recommender Syst., 2013, pp. 16–23.
[10] L. Kolb, A. Thor, and E. Rahm, “Parallel sorted neighborhood blocking with MapReduce,” in Proc. Conf. Datenbanksysteme in Buro, Technik und Wissenschaft, 2011.