Volume 6, Issue 11, November 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Recognizing and Removing Redundancy from the Distributed Reliable Database

[1]P. Vijayakumar [2]Dr. G. T. Shrivakshan
[1] M.Phil., Research Scholar, [2] MCA., M.Phil., Ph.D., Head of PG and Research Department
[1, 2] Department of Computer Science, Swami Vivekananda Arts and Science College, Villupuram. Thiruvalluvar University, Vellore, TamilNadu, India

Abstract—Duplicate detection is the process of identifying multiple representations of same real world entities. Today, duplicate detection methods need to process ever larger datasets in ever shorter time: maintaining the quality of a dataset becomes increasingly difficult. We present 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 our progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work. Aiming to address the above security challenges, this project makes the first attempt to formalize the notion of distributed reliable deduplication system. Security analysis demonstrates that our deduplication systems are secure in terms of the definitions specified in the proposed security model. As a proof of concept, we implement the proposed systems and demonstrate that the incurred overhead is very limited in realistic environments.

Index Terms—Duplicate Detection, Data set, Data cleaning, Reliable data, Redundancy

I. INTRODUCTION

In computing, data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data. Related and somewhat synonymous terms are intelligent (data) compression and single-instance (data) storage. This technique is used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent. In the deduplication process, unique chunks of data, or byte patterns, are identified and stored during a process of analysis. As the analysis continues, other chunks are compared to the stored copy and whenever a match occurs, the Redundant chunk is replaced with a small reference that points to the stored chunk. Given that the same byte pattern may occur dozens, hundreds, or even thousands of times (the match frequency is dependent on the chunk size), the amount of data that must be stored or transferred can be greatly reduced.

This type of deduplication is different from that performed by standard file-compression tools, such as LZ77 and LZ78. Whereas these tools identify short repeated substrings inside individual files, the intent of storage-based data deduplication is to inspect large volumes of data and identify large sections – such as entire files or large sections of files – that are identical, in order to store only one copy of it. This copy may be additionally compressed by single-file compression techniques. For example a typical email system might contain 100 instances of the same 1 MB (megabyte) file attachment. Each time the email platform is backed up, all 100 instances of the attachment are saved, requiring 100 MB storage space. With data deduplication, only one instance of the attachment is actually stored; the subsequent instances are referenced back to the saved copy for deduplication ratio of roughly 100 to 1.

Progressive duplicate detection identifies most duplicate pairs early in the detection process. Instead of reducing the overall time needed to finish the entire process, progressive approaches try to reduce the average time after which a duplicate is found. Early termination, in particular, then yields more completes results on a progressive algorithm than on any traditional approach. Databases play an important role in today's IT based economy. Many industries and systems depend on the accuracy of databases to carry out operations.

Therefore, the quality of the information stored in the databases, can have significant cost implications to a system that relies on information to function and conduct business. In an error-free system with perfectly clean data, the construction of a comprehensive view of the data consists of linking-in relational terms, joining two or more tables on their key fields. Unfortunately, data often lack a unique, global identifier that would permit such an operation. Furthermore, the data are neither carefully controlled for quality nor defined in a consistent way across different data sources. Thus, data quality is often compromised by many factors, including data entry errors (e.g., student instead of student), missing integrity constraints (e.g., allowing entries such as Employee Age=567), and multiple conventions for recording information To make things worse, in independently managed databases not only the values, but the structure, semantics and underlying assumptions about the data may differ as well.

Progressive duplicate detection identifies most duplicate pairs early in the detection process. Instead of reducing the overall time needed to finish the entire process, progressive approaches try to reduce the average time after which a

duplicate is found. Progressive techniques make this trade-off more beneficial as they deliver more complete results in shorter amounts of time. Progressive Sorted Neighbourhood method take clean dataset and find some duplicate records and Progressive Blocking take dirty datasets and detect large duplicate records in databases.

II. RELATED WORK

Much research on duplicate detection also known as entity resolution and by many other names, focuses on pair-selection algorithms that try to maximize recall on the one hand and efficiency on the other hand. The most prominent algorithms in this area are Blocking and the Sorted neighbourhood Method.

Adaptive Techniques. Previous publications on duplicate detection often focus on reducing the overall runtime. Thereby, some of the proposed algorithms are already capable of estimating the quality of comparison candidates. The algorithms use this information to choose the comparison candidates more carefully. For the same reason, other approaches utilize adaptive windowing techniques, which dynamically adjust the window size depending on the amount of recently found duplicates. These adaptive techniques dynamically improve the efficiency of duplicate detection, but in contrast to our progressive techniques, they need to run for certain periods of time and cannot maximize the efficiency for any given time slot.

Progressive Techniques. In the last few years, the economic need for progressive algorithms also initiated some concrete studies in this domain. For instance, ay-as-you-go algorithms for information integration on large scale datasets have been presented. Other works introduced progressive data cleansing algorithms for the analysis of sensor data streams. However, these approaches cannot be applied to duplicate detection.

Xiao et al. proposed a top-k similarity join that uses a special index structure to estimate promising comparison candidates. This approach progressively resolves duplicates and also eases the parameterization problem. Although the result of this approach is similar to our approaches (a list of duplicates almost ordered by similarity), the focus differs: Xiao et al. find the top-k most similar duplicates regardless of how long this takes by weakening the similarity threshold; we find as many duplicates as possible in a given time. That these duplicates are also the most similar ones is a side effect of our approaches.

Indexing techniques for scalable record linkage and deduplication

With the increasing size of today's databases, the complexity of the matching process becomes one of the major challenges for record linkage and deduplication. In recent years, various indexing techniques have been developed for record linkage and deduplication. They are aimed at reducing the number of record pairs to be compared in the matching process by removing obvious nonmatching pairs, while at the same time maintaining high matching quality. Record matching, which identifies the records that represent the same real-world entity, is an important step for data integration. Most state of the art record matching methods a supervised, which requires the user to provide training data. In this paper, the necessity of comparing records in a file with those in other file in an effort to find out which pair of records relate to the same data unit which arises many contexts most of which can be categorized as either the same record in main file storage.

Generalization of blocking and windowing for duplicate detection

The first difficulty is the quadratic nature of the problem: Conceptually, each candidate must be compared with every other candidate. To approach this problem and thus improve efficiency of duplicate detection, many solutions have been proposed to reduce the number of comparisons. The general idea is to avoid comparisons of vastly different objects and to concentrate on comparisons of objects that have at least some quickly detectable similarity. We present a comparison and generalized model of the well-known locking and windowing methods and propose a generalized algorithm, the Sorted Blocks method, with a parameter whose extreme settings yield the two methods, respectively. An experimental evaluation for precision and recall compares the two methods and find an optimal setting for the generalized method.

Deduplication overview

Deduplication may occur "in-line", as data is flowing, or "post-process" after it has been written.

Post-process deduplication

With post-process deduplication, new data is first stored on the storage device and then a process at a later time will analyse the data looking for duplication. The benefit is that there is no need to wait for the hash calculations and lookup to be completed before storing the data thereby ensuring that store performance is not degraded. Implementations offering policy-based operation can give users the ability to defer optimization on "active" files, or to process files based on type and location. One potential drawback is that you may unnecessarily store duplicate data for a short time which is an issue if the storage system is near full capacity.

In-line deduplication

This is the process where the deduplication hash calculations are created on the target device as the data enters the device in real time. If the device spots a block that it already stored on the system it does not store the new block, just references to the existing block. The benefit of in-line deduplication over post-process deduplication is that it requires less storage as data is not duplicated. On the negative side, it is frequently argued that because hash calculations and

lookups takes so long, it can mean that the data ingestion can be slower thereby reducing the backup throughput of the device. However, certain vendors with in-line deduplication have demonstrated equipment with similar performance to their post-process deduplication counterparts. Post-process and in-line deduplication methods are often heavily debated.

Source versus target deduplication

Another way to think about data deduplication is by where it occurs. When the deduplication occurs close to where data is created, it is often referred to as "source deduplication". When it occurs near where the data is stored, it is commonly called "target deduplication".

III. EXISTING WORK

In existing work, however, we focus on progressive algorithms, which try to report most matches early on, while possibly slightly increasing their overall runtime. To achieve this, they need to estimate the similarity of all comparison candidates in order to compare most promising record pairs first. We propose two novel, progressive duplicate detection algorithms namely progressive sorted neighborhood method (PSNM), which performs best on small and almost clean datasets, and progressive blocking, which performs best on large and very dirty datasets. Both enhance the efficiency of duplicate detection even on very large datasets.

We introduce a concurrent progressive approach for the multi-pass method and adapt an incremental transitive closure algorithm that together forms the first complete progressive duplicate detection workflow. We define a novel quality measure for progressive duplicate detection to objectively rank the performance of different approaches. We exhaustively evaluate on several real-world datasets testing our own and previous algorithms.

Progressive duplicate detection identifies most duplicate pairs early in the detection process. Instead of reducing the overall time needed to finish the entire process, progressive approaches try to reduce the average time after which a duplicate is found.

Then the progressive algorithm discovers more duplicate pairs at t than the corresponding traditional algorithm. Typically, t is smaller than the overall runtime of the traditional algorithm.

Limitations

Data reliability is actually a very critical issue in a deduplication storage system because there is only one copy for each file stored in the server shared by all the owners.

Most of the previous deduplication systems have only been considered in a single-server setting. The traditional deduplication methods cannot be directly extended and applied in distributed and multi-server systems.

IV. PROPOSED METHODOLOGY

Here propose two dynamic progressive duplicate detection algorithms, PSNM and PB with NDD (Near Duplicate Detection), which expose different strengths and outperform current approaches. We introduce a concurrent progressive approach for the multi-pass method and adapt an incremental transitive closure algorithm that together form the first complete progressive duplicate detection workflow.

We define a novel quality measure for progressive duplicate detection to objectively rank the performance of different approaches. We exhaustively evaluate on several real-world datasets testing our own and previous algorithms. The duplicate detection workflow comprises the three steps pair-selection, pair-wise comparison, and clustering. For a progressive workflow, only the first and last step need to be modified.

Therefore, we do not investigate the comparison step and propose algorithms that are independent of the quality of the similarity function. Our approaches build upon the most commonly used methods, sorting and (traditional) blocking, and thus make the same assumptions: duplicates are expected to be sorted close to one another or grouped in same buckets, respectively.

Advantages

Distinguishing feature of our proposal is that data integrity, including tag consistency, can be achieved. To our knowledge, no existing work on secure deduplication can properly address the reliability and tag consistency problem in distributed storage systems. Our proposed constructions support both file-level and block-level deduplications. Security analysis demonstrates that the proposed deduplication systems are secure in terms of the definitions specified in the proposed security model. In more details, confidentiality, reliability and integrity can be achieved in our proposed system. We implement our deduplication systems using the Ramp secret sharing scheme that enables high reliability and confidentiality levels. Our evaluation results demonstrate that the new proposed constructions are efficient and the redundancies are optimized and comparable with the other storage system supporting the same level of reliability.

NDD Algorithm Pseudo code:

Initialize the User U_j id: IDU_j j=1….N (No.of users).
File identifier: F_{idj} j= 1….N (No. of files).
Blocks of files: B_i i=1… N (No. of blocks)
For each block B_i
For each file F_{idj}
Compute hash value of the file F_{id}

Stores the F_{id1} as H (User Id | File Name) to the multiple servers
Obtain the unique Block identifier B_{id} for each file in the server.
End for End for
User $U_{j\ new}$ is found,
Compute the F_{idj}, B_{idj} for the Server j
If (present value F_{idj}, B_{idj} $\neq$ old value F_{idj}, B_{idj})
The file is stored to the server
Else
Detects the duplicated blocks of a file.

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 attributes 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 neighbourhood method. When using early termination, this parameter can be set to an optimistically high default value. Let us consider, it has the default value 1. The last parameter N specifies the number of records in the dataset. This number can be gleaned in the sorting step, but listed as a parameter for presentation purposes.

V. SYSTEM ARCHITECTURE

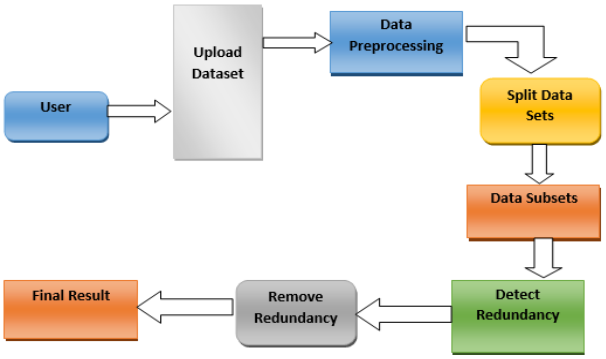


Fig 1: System Architecture

VI. EXPERIMENTAL RESULTS

We consider restaurant information from an internal operational data warehouse and introduce errors. Because we start from real data all characteristics of real data: variations in the lengths of strings, numbers of tokens in and frequencies of attribute values, co-occurrence patterns, etc. are preserved. Since, we know the duplicate tuples and their correct counterparts in the erroneous dataset; we can evaluate duplicate elimination algorithms.

The execution bottleneck for duplicate detection is commonly the attribute correlation with likeness measures between the record sets which is quite expensive one. To dodge this restrictively expensive analysis of all sets of records, a basic method is to precisely segment the records into smaller subsets and quest for copies just inside of these allotments. Two contending methodologies are regularly referred to: Blocking techniques allotment records into disjoint subsets, for occurrence utilizing zip code as apportioning key. Sorted-neighbourhood based strategies that sort the information as per some key, for example, last name, and afterward slide a window of altered size over the sorted information and look at sets just inside of the window.

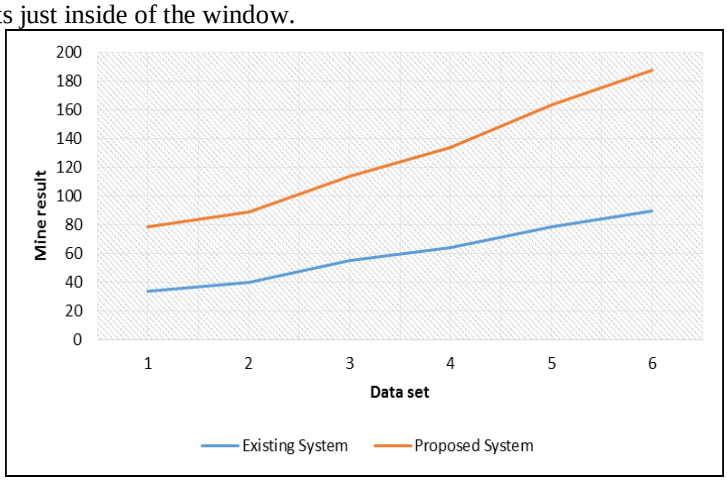
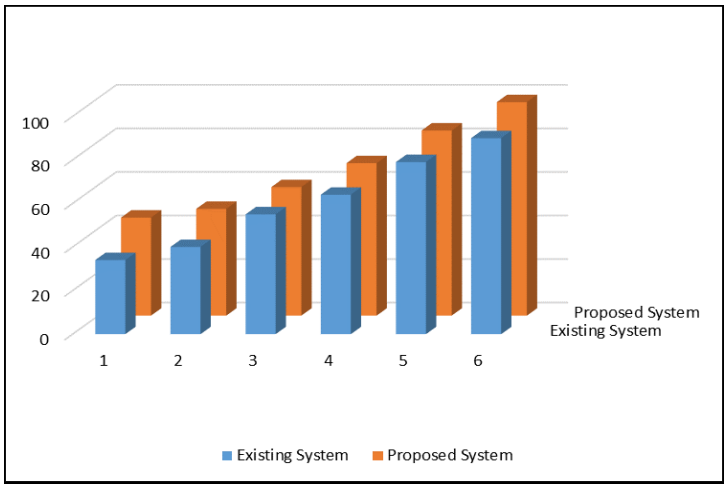


Fig 2. Based on the effectiveness metric for our proposed work with existing work.

VII. PERFORMANCE RESULT

To run the duplicate detection, here use a Dell PowerEdge R620 with two Intel E5-2650 2.00 GHz CPUs and 128 GB DDR3-1600 RAM. Note that although the server provides 16 cores, the current implementations of all algorithms are single-threaded and, therefore, utilize only one core. Hence, all algorithms can further be improved by

parallelization. The server's main memory of 128 GB can hold 15 million records of the given plista-dataset, which leads to seven partitions overall. Due to the size of the dataset and the high number of expected duplicates, we also increase the maximum window size to 50 for the SNM-approaches and the block size to 6 and maximum block range to 8 for the PB algorithm.



VIII. CONCLUSION WITH FUTURE WORK

In this paper, we exploited block-level deduplication algorithm to develop a high quality, scalable, and efficient algorithm for detecting progressive duplicates in dimensional data. Our approach is able to automatically suggest duplication functions based on evidence present in the data repositories. The suggested functions properly combine the best evidence available in order to identify whether two or more distinct record entries are replicas (i.e., represent the same real-world entity) or not. This is extremely useful for the no specialized user, who does not have to worry about setting up the best set of evidence for the replica identification task. In future work, we want to combine our progressive approaches with scalable approaches for duplicate detection to deliver results even faster.

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] 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.

[5] 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.

[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 28th Int. Conf. Data Eng., 2012, pp. 1073–1083.

[10] S. Yan, D. Lee, M.-Y. Kan, and L. C. Giles, "Adaptive sorted neighbourhood methods for efficient record linkage," in Proc. 7th ACM/ IEEE Joint Int. Conf. Digit. Libraries, 2007, pp. 185–194.

