International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-1 Number-2 Issue-2 December 2011

Result Analysis and Benefits of Detecting Replicate Documents Using MD5 Hash Function

Pushpendra Singh Tomar[1], Maneesh Shreevastava[2]
M-Tech (IT), LNCT, Bhopal[1]
Head and Professor, IT, LNCT Bhopal[2]

Abstract

The definition of what constitutes a replicate has somewhat different interpretations. For instance, some define a replicate as having the exact syntactic terms and sequence, whether having formatting differences or not. In effect, there are either no difference or only formatting differences and the contents of the data are exactly the same. In any case, data replication happens all the time. In large data warehouses, data replication is an inevitable phenomenon as millions of data are gathered at very short intervals. In this paper we provide a detail result analysis on the basis of our approach and the previous one.

Keywords

Replication, Result, MD5, documents

1. Introduction

The definition of what constitutes a replicate has somewhat different interpretations. For instance, some define a replicate as having the exact syntactic terms and sequence, whether having formatting differences or not. In effect, there are either no difference or only formatting differences and the contents of the data are exactly the same. In any case, data replication happens all the time. In large data warehouses, data replication is an inevitable phenomenon as millions of data are gathered at very short intervals.

Data warehouse involves a process called ETL which stands for extract, transform and load. During the extraction phase, multitudes of data come to the data warehouse from several sources and the system behind the warehouse consolidates the data so each separate system format will be read consistently by the data consumers of the warehouse. Data portals are everywhere. The tremendous growth of the Internet has spurred the existence of data portals for nearly every topic. Some of these portals are of general interest; some are highly domain specific. Independent of the focus, the vast majority of the portals obtain data, loosely called documents, from multiple sources [1].

Obtaining data from multiple input sources typically results in replication. The detection of replicate documents within a collection has recently become an area of great interest [2] and is the focus of our described effort. Typically, inverted indexes are used to support efficient query processing in information search and retrieval engines. Storing replicate documents affects both the accuracy and efficiency of the search engine. Retrieving replicate documents in response to a user's query clearly lowers the number of valid responses provided to the user, hence lowering the accuracy of the user's response set. Furthermore, processing replicates necessitates additional computation Replicates are abundant in short text databases. For example, popular mobile phone messages may be forwarded by millions of people, and millions of people may express their opinions on the same hot topic by mobile phone messages. In our investigation on mobile phone short messages, more than 40% short messages have at least one exact replicate. An even larger proportion of short messages are near-replicates. Detecting and eliminating these replicate short messages is of great importance for other short text processing, such as short text clustering, short text opinion mining, short text topic detection and tracking, short message community uncovering. Exact replicate short texts are easy to identify by standard hashing schemes. Informal abbreviations without introducing any additional benefit. Hence, the processing efficiency of the user's query is lowered. A problem introduced by the indexing of replicate documents is potentially skewed collection statistics. Collection statistics are often used as part of the similarity computation of a query to a document. Hence, the biasing of collection statistics may affect the overall precision of the entire system.

Simply put, not only is a given user's performance compromised by the existence of replicates, but also the overall retrieval accuracy of the engine is jeopardized. The definition of what constitutes a replicate is unclear. For instance, a replicate can be

defined as the exact syntactic terms, without formatting differences. Throughout our efforts however, we adhere to the definition previously referred to as a measure of resemblance [3]. The general notion is that if a document contains roughly the same semantic content it is a replicate whether or not it is a precise syntactic match. When searching web documents, one might think that, at least, matching URL's would identify exact matches. However, many web sites use dynamic presentation wherein the content changes depending on the region or other variables. In addition, data providers often create several names for one site in an attempt to attract users with different interests or perspectives. For instance, Fox4, Onsale-Channel-9, and Real-TV all point to an advertisement for real TV.

2. Clustering Analysis

Litigators and investigators are reaching the limits of their ability to process the immense amount of information generated by electronic communications involved in complex matters. As a result, some firms are using clustering analysis and data visualization technology to help them greatly streamline the review stage of e-discovery.

Clustering analysis technologies automatically group documents based on relationships and "concepts" (related words or phrases) in the data, providing a more robust view of the contents than keyword searches alone. By using the technology to cluster related documents and visualize them in a "concept map" interface, attorneys can tag groups of highly relevant documents while quickly setting aside irrelevant documents.

For example, attorneys for a large financial institution facing class-action litigation recently used clustering to rapidly reduce a population of 550,000 documents to 7,000 documents that merited a more detailed privileged review saving substantial time and money for their client.

3. Elimination of exact replicates

In this Section, we describe the data model and algorithms involved in the replicates elimination mechanism. The data model relies on 3 main classes: instance, volume and block.

The instance class provides a centralized view of a storage space composed of volumes containing blocks. Each block keeps a document and related operational meta-data. The signature is the number obtained from applying a fingerprinting algorithm to the document. A content key contains the signature of the document and the volume where it was stored. A block holds a unique document within the volume.
It is composed by a header and a data container (Figure 1)

1). The data container keeps the document. The header contains information about the software version, the document's original size in bytes and a reference counter that keeps track of the difference between the storage and deletes requests performed on the document, allowing independent applications to share the same instance without interfering with each other's data processing. The header also specifies the algorithm used to compress the document, allowing the coexistence of several compression types within the volume and the application of suitable algorithms according to the document's format. The storage structure of a volume is a tree containing blocks on its leafs. Figure 1 illustrates a storage structure with depth 3. The nodes within each level of depth are identified by numbers represented in hexadecimal format from 0 to FF. The tree depth can change within the volumes that compose an instance, according to the storage capacity of the node.

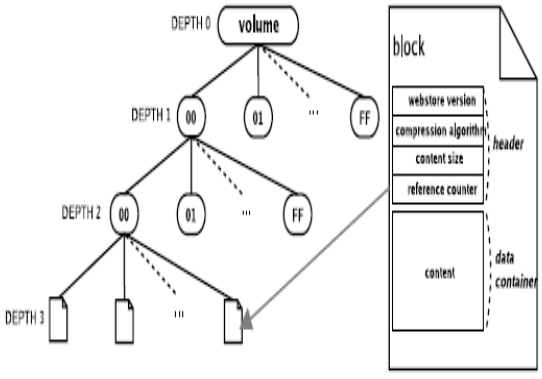

Figure 1: Storage structure of a volume: a tree holding blocks on the leafs.

The location of a block within the volume tree is obtained by applying a function called signlocation to the document's signature. Assuming that the signature of a document is unique, two documents have the same location within a volume if they are replicates. Consider a volume tree with depth n and a signature with m bytes of length. Signlocation uses the (n - 1) most significant bytes in the signature to

identify the path to follow in the volume tree. The ith byte of the signature identifies the tree node with depth i. The remaining bytes of the signature (m-n-1) identify the block name on the leaf of the tree. For instance, considering a volume tree with depth 3, the block holding content with signature ADEE2232AF3A4355 would be found in the tree by following the nodes AD, EE and leaf 2232AF3A4355.

The detection of replicates is performed during the storage of each document, ensuring that each distinct document is stored in a single block within the instance. When a client requests the storage of a document, the system performs a sequence of tasks:

1. Generates a signature s for the document;
2. Applies sign location to the signature and obtains the location l of the corresponding block;
3. Searches for a block in location l within the n volumes that compose the instance, multicasting requests to the volumes;
4. If a block is found on one of the volumes, the document is considered to be a duplicate and its reference counter is incremented. Otherwise, the document is stored in a new block with location l in the volume identified by s mod n;
5. Finally, a content key referencing the block is returned to the client.

4. Fake Replication

Theoretically, if two documents have the same signature they are replicates. However, fingerprinting algorithms present a small probability of collision that causes the generation of the same signature for two different documents.

We believe that the probability of losing a document due to a disk error or bug on the underlying software (e.g imported software libraries or hardware drivers) is bigger than the probability of fingerprint collisions. Nevertheless, we support 3 modes for the store operation to fulfill the requirements of applications that may need absolute certainty that fake replicates do not occur: force-new, regular and compare. When using the force-new mode, the elimination of replicates is switched off and a new block is created to store each document. This semantic is useful if one knows that the collection does not contain replicates. The regular mode (default) detects a collision if two contents have the same signature but different sizes.

In this case, an overflow block is created to keep the document. However, the success of this heuristic depends on the distribution of the document sizes and collisions will not be detected among documents with the same size. We computed the distribution of sizes for a random sample of 3.2 million web pages and found that the probability of two random web pages having the most frequent size. Assuming that the probability of two pages having the same size and the probability of fingerprint collision between them are independent events, our results indicate that the comparison of sizes can substantially reduce the occurrence of fake replicates without requiring longer fingerprint signatures or additional meta-data. The compare mode relies on size and byte wise comparison of the documents to detect collisions.

If two contents have the same signature but different sizes, or have the same size but are not byte equal, a collision is detected. Fake replicates never occur when using this store mode.

Replication Concept

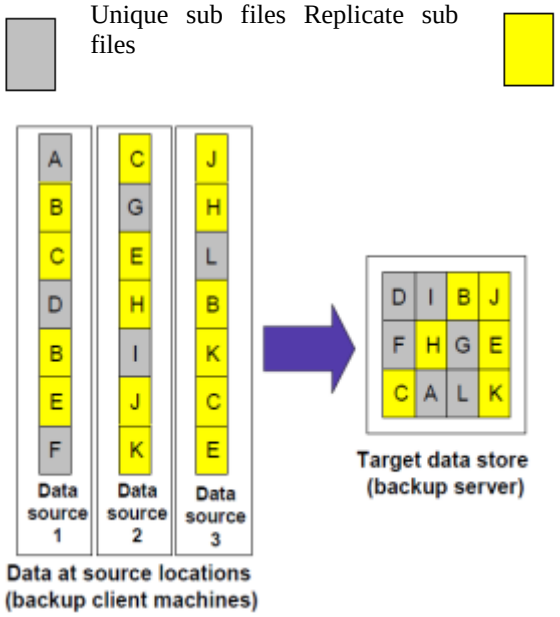

Figure 2: The Replicate Concept

5. Result Analysis

Unfortunately, there is no available absolute body of truth or a benchmark to evaluate the success of these techniques. Thus, it is difficult to get any type of quantitative comparison of the different algorithms and thresholding techniques. This is not likely to change in the near future. As document collections grow, the likelihood of judgments of Replicates being made is small; therefore, the best that can be hoped for is to provide fast efficient techniques for

duplication detection that can be passed on to analysis for further evaluation.

The most obvious way to identify Replicate documents is to directly hash the entire contents of a document to a unique value. This type of approach finds exact matches by comparing the calculated hash value with the other document hash values for Replicate document detection. However, they are used to see if a particular document has changed. We experimented with various filtration techniques to improve the resilience of the direct hash approach to small document changes. If a simple filtration technique based on strictly syntactic information is successful then fast Replicate and similar document detection could be achieved.

Table 1. Unique Documents and Percent Found as Replicate for Small, Html, Image, Audio, Video and other File

File Type	Percent Found as Replicates	Unique Documents Found in Collection
Small File	6 %	17906
Html File	2.22 %	2423
Picture File	8 %	227654
Audio File	6 %	8743
Video File	4 %	5462
Other File	6.22 %	50112

The effect of filtering tokens on the degree of Replicate document detection is shown in Table 1. We used the LNCT collection because the collection is fully replicated. Therefore, the percentage of Replicates found is an evaluation metric of the effectiveness of the filter. Also shown in the table, is the percentage of terms retained after each filtering technique. Generally speaking, as we show in Table 1, the higher the filtration, the greater the degree of detection.

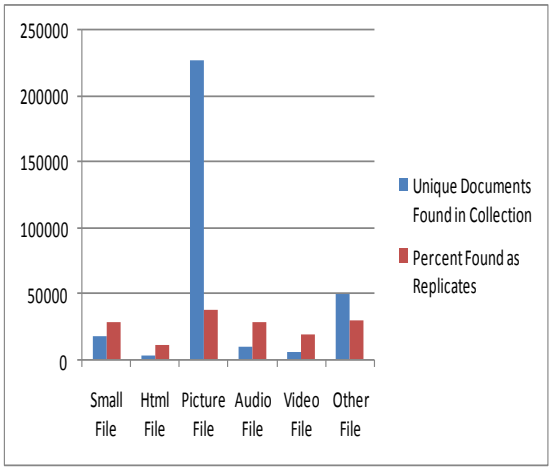

Figure 3: Unique Documents and Percent Found as Replicate

While several of the filtration techniques do find 88% of the collection, the Replicates they find are near or exact matches and a maximum number of unique documents of 92038. In contrast, F-Replicate for this same collection detects 96.2% duplication and a maximum number of unique documents of 87568. Clearly the lower the maximum number of unique documents, the better is the detection capability.

Table 2 Show Computing Result with Time Elapse and File Size

Total files found	Sum of file sizes (MB)	Elapsed Time for search (Seconds)	Computing hashes files	Sum of file sizes is now (MB)	Elapsed Time for computing hashes (Seconds)	deleted files	Elapsed Time for Delete (Seconds)
6	0.00	0.29777	2	0.00	0.49986	1	0.7822
64	7.97	0.12056	23	0.13	0.22485	22	0.45414
84	8.85	0.094894	54	1.82	0.20726	53	0.44475
609	1643.07	0.362	82	1.38	0.46681	81	0.72283
635	212.29	0.35789	374	1.01	0.47502	373	0.77107
910	215.36	1.3129	311	17.53	1.4202	310	1.6936
934	69.23	1.2385	144	6.74	1.3548	143	1.5976
4869	2972.13	3.3627	2099	191.37	3.5272	2098	4.069
74077	710.64	29.7559	66896	146.02	31.0575	66895	45.7813
77236	1661.56	32.038	69303	266.29	33.733	69301	64.0408

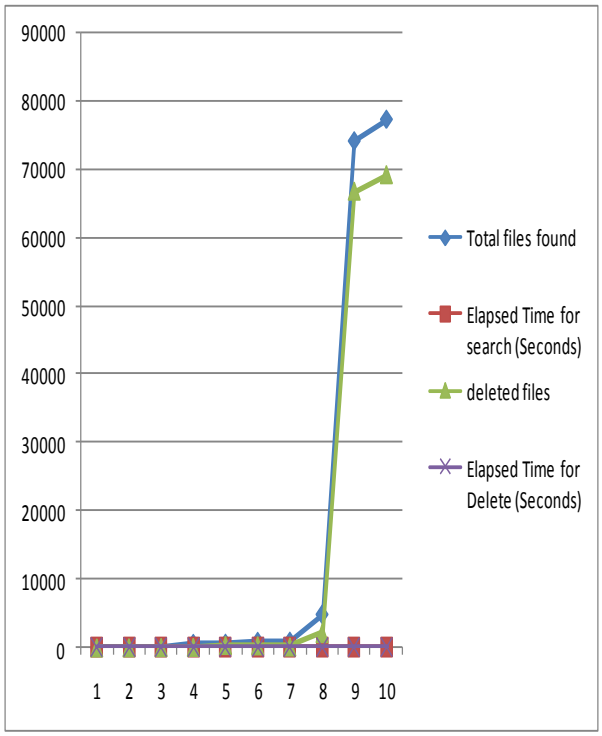

Figure 4 Detecting Total File and Deleted File With Time Elapsed

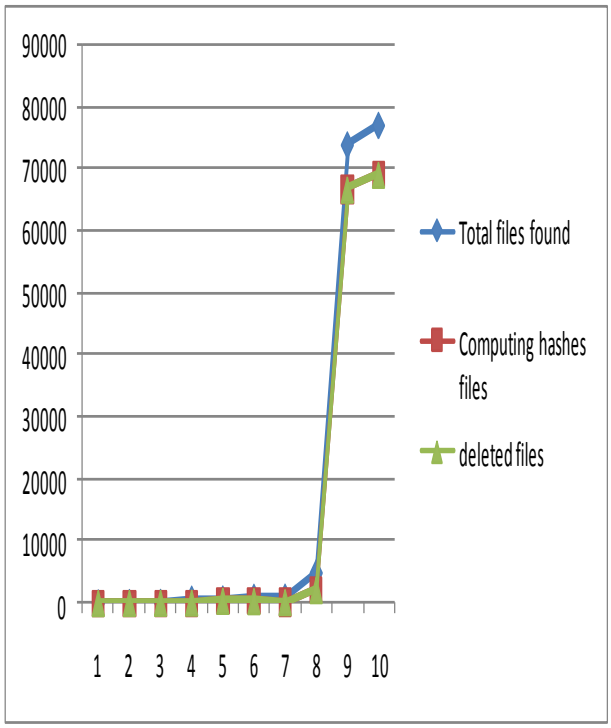

Figure 6 Total Documents with Computing Hashes File and Deleted File

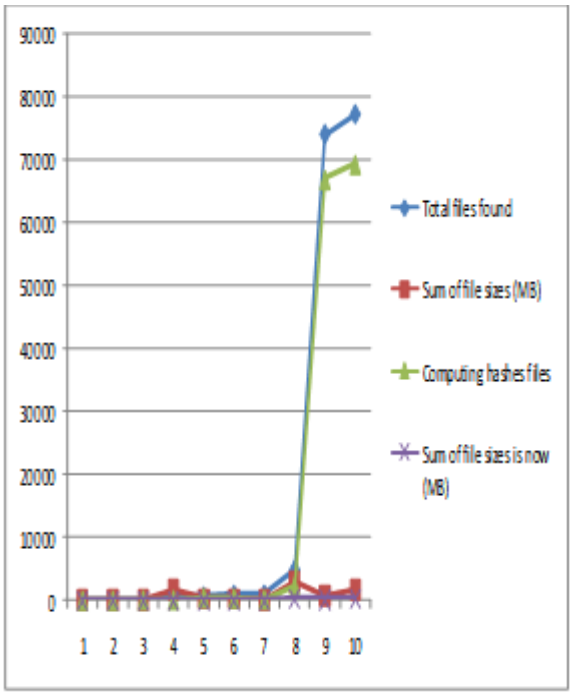

Figure 5 Total and Computing Hashes File With Size of File in MB

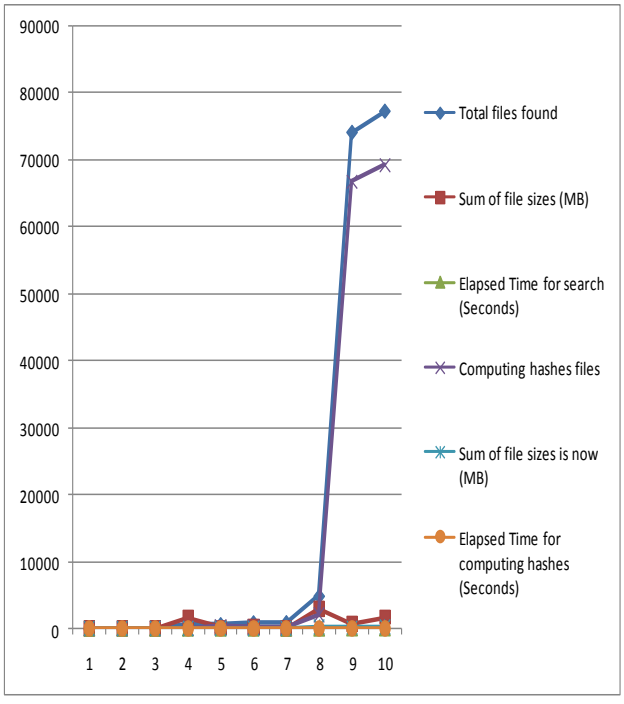

Figure 7 Detecting Total and Similar Documents with Time Elapsed and Size of File in MB

Our simple filtering techniques reduced the list of tokens used to create the hash. By eliminating white spaces and only keeping unique tokens, many small document changes are eliminated. Keeping only unique tokens eliminates movement of paragraph errors, stemming removes errors caused by small token changes, and stop word removal removes errors caused by adding or removing common irrelevant tokens, in terms of semantics. We found that removing tokens containing 'special characters' (i.e.=, -,D, etc.) performed the best in terms of removing tokens from documents.

Algorithms for detecting similar documents are critical in applications where data is obtained from multiple sources. The removal of similar documents is necessary, not only to reduce runtime, but also to improve search accuracy. Today, search engine crawlers are retrieving billions of unique URL's, of which hundreds of millions are Replicates of some form. Thus, quickly identifying Replicate detection expedites indexing and searching. One vendor's analysis of 1.2 billion URL's resulted in 400 million exact Replicates found with a MD5 hash. Reducing the collection sizes by tens of percentage points results in great savings in indexing time and a reduction in the amount of hardware required to support the system. Last and probably more significant, users benefit by eliminating Replicate results. By efficiently presenting only unique documents, user satisfaction is likely to increase.

We proposed a new similar document detection algorithm called F-Replicate and evaluated its performance using multiple data collections. The document collections used varied in size, degree of expected document duplication, and document lengths. The data was obtained from LNCT, Server and from Home PC. F-Replicate relies on collection statistics to select the best terms to represent the document. F-Replicate was developed to support web document collections.

Thus, unlike many of its predecessors, F-Replicate efficiently processes large collections and does not neglect small documents. In comparison to the prior state of threat, In terms of human usability, no similar document detection approach is perfect however; our experimentation shows the F-Replicate to be the most effective approach for finding Replicate documents. The ultimate determination of how similar a document must be to be considered a Replicate, relies on human judgment. Therefore, any solution must be easy to use. To support ease of use, all potential Replicates should be uniquely grouped together.

6. Conclusion

We proposed a new replicate document detection algorithm called DRD and evaluated its performance using multiple data collections. The document collections used varied in size, degree of expected document replication, and document lengths. In terms of human usability, no similar document detection approach is perfect. The ultimate determination of how similar a document must be to be considered a replicate relies on human judgment. Therefore, any solution must be easy to use. To support ease of use, all potential replicates should be uniquely grouped together.

Therefore, any match in even single results in a potential replicate match indication. This results in the scattering of potential replicates across many groupings, and many false positive potential matches. DRD, in contrast, treats a document in its entirety and maps all potential replicate s into a single grouping. This reduces the processing demands on the user. This paper has been felt necessary when the work on developing Replicate document detection is very hopeful, and is still in promising status. This survey paper intends to aid upcoming researchers in the field of Replicate document detection in web crawling to understand the available methods and help to perform their research in further direction.

References

[1] Broder, A., Glassman, S., Manasse, S., and Zweig, G. 1997. Syntactic clustering of the web. In Proceedings of the Sixth International World Wide Web Conference (WWW6'97) (Santa Clara, CA., April). 391–404.

[2] Shivakumar, N. and Garica-Molina, H. 1998. Finding near-replicas of documents on the web. In Proceedings of Workshop on Web Databases (WebDB'98) (Valencia, Spain, March). 204–212.

[3] HEINTZE, N. 1996. Scalable document fingerprinting. In Proceedings of the Second USENIX Electronic Commerce Workshop (Oakland, CA., November). 191–200.