Query-biased Near Duplicate Web Document Detecting: Effective, Efficient and Customizable

Bingfeng Pi, Shunkai Fu, Gang Zou, Jia Guo and Song Han

***Abstract*—Near-duplicate web documents are abundant and annoying to both search service providers and end-users. Two such documents share the same primary content, but differ from each other with a minor portion where, for instance, advertisements or portal information are presented. In the course of developing a near-duplicate detection system for a large document repository, we make three contributions. First, we demonstrate that Charikar's SimHash is appropriate for measuring the similarities between documents. Secondly, we reduce the search space by checking only among the results retrieved by queries, which can be quite time and memory efficient and effective if works with search log meanwhile. Thirdly, which duplicated documents to remove are determined given their authority scores determined by PageRank algorithm. General PC can afford such processing considering that the scale of documents returned by a specific query is much smaller compared with the whole repository. We also discuss the customizability of such a query-biased approach, e.g. different k parameter (in SimHash) for queries with different popularities as shown by the search log. Besides, the processing can proceed in a parallel way.**

I. INTRODUCTION

DUPLICATE and near-duplicate web documents are posing large problems on web search engines: They increase the space required to store the index, slow down serving results, and annoy the users. Thus, the algorithms for detecting and deleting these documents, effectively and efficiently, are highly expected today [1, 4].

Documents that are exact duplicates of each other, e.g. due to mirroring [6] and plagiarism [7], are easy to identify by standard check-summing techniques, like MD5 (Message-Digest algorithm 5) [8]. Compared with it, the identification of near-duplicate documents is more difficult a problem. Two documents are near duplicated if they are identical in terms of content but differ in a small portion of the document such as portal information, advertisement, counters and timestamps. Fig 1 is such an example where two pages have same primary content but small difference in the highlighted rectangle area. Elimination of near duplicates saves network bandwidth, reduces storage costs and improves the quality of search indexes.

Bingfeng Pi: Roboo Inc.(Leading mobile search engine in China), F1-13, No. 171, Jin Ji Hu Road, Suzhou Industiral Park, Suzhou, Jiangsu Province, P.R.China 215021

Shunkai Fu: Roboo Inc. and Ecole Polytechnique de Montreal, Montreal, Quebec, Canada (email: shukai.fu@polymtl.ca).

Gang Zou: Roboo Inc.

Jia Guo: Roboo Inc.

Song Han: Ph.D, CEO of Roboo Inc (email: song.han@roboo.com)

Compared with finding exact duplicates, a system for the detection of near-duplicate documents faces a larger and more complex search space in which both duplicates and near-duplicates are deemed to process, and the determination of near-duplicate documents are known as more challenging. An ideal solution is expected to not only help us find both duplicates and near duplicates, but also work in an efficient manner.

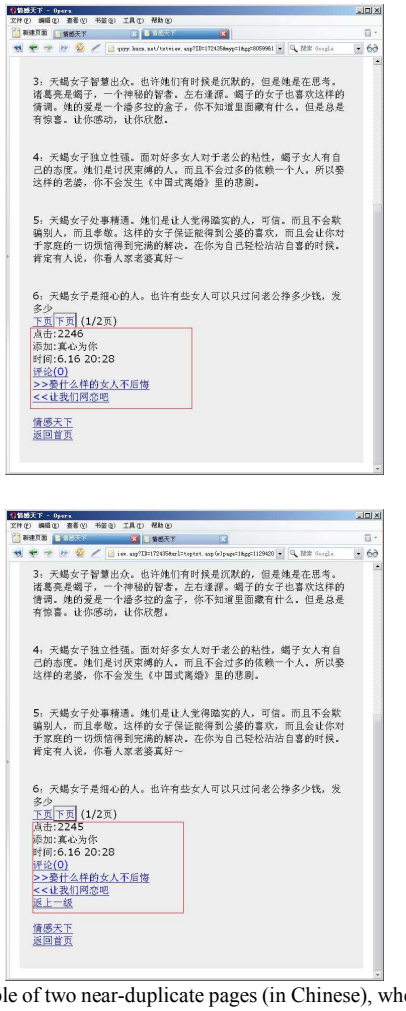

Fig. 1. An example of two near-duplicate pages (in Chinese), where there are minor differences exist in the highlighted area.

Our contributions include three aspects. Firstly, we demonstrate that Charikar's SimHash [2] technique is effective and appropriate to measure the similarity between documents, although we are not the first to recognize this. Secondly, we propose to search for near-duplicate documents among the retrieved results of queries. This query-biased

approach allows us to divide the whole document repository into much smaller sub-spaces so that the intensive computing resource required for searching duplicates from a large repository becomes affordable, even on general PC. It is shown perform well if work with the search log from which queries and their corresponding frequencies (times of being searched) can be extracted. Besides, this method makes possible parallel processing, and k (an important parameter in Charikar's method that determines the degree of duplicate) can be customized flexibly as well. Thirdly, we determine which duplicated documents to be deleted with a reference to their corresponding PageRank[5] scores considering that is a query independent, fair and trustable measure.

The paper is organized as follows: Section 2 describes the algorithm in detail. Section 3 presents the experiments and the evaluation results. We conclude in Section 4.

II. DESCRIPTION OF THE ALGORITHMS

A. Fingerprinting with SimHash and Hamming Distance

Charikar's SimHash [2], actually, is a fingerprinting technique that produces a compact sketch of the objects being studied, no matter documents discussed here or images. So, it allows for various processing, once applied to original data sets, to be done on the compact sketches, a much smaller and well formatted (fixed length) space. With documents, SimHash works as follows: a web document is converted into a set of features, each feature tagged with its weight. Then, we transform such a high-dimensional vector into an $f-bit$ fingerprint where f is quite small compared with the original dimensionality. An excellent comparison of SimHash and the traditional Broder's shingle-based fingerprints [3] can be found in Henzinger [4].

To make the document self contained, here, we give the algorithm's specification in Fig 2, and explain it with a little more detail. We assume the input, document D, is pre-processed and composed with a series of features (tokens). Firstly, we initialize an f-dimensional vector V with each dimension as zero (line 1). Then, for each feature, it is hashed into an $f-bit$ hash value. These f bits increment or decrement the f components of the vector by the weight of that features based on the value of each bit of the hash value calculated (line 4-8). Finally, the signs of the components determine the corresponding bits of the final fingerprint (line 9-11).

SimHash has two important but somewhat conflicting properties: (1) the fingerprint of a document is a "hash" of its features, and (2) Similar documents have similar hash values. The latter property is quite different from traditional hash function, like MD5 or SHA-1 (Secure Hash Algorithm), where the hash-values of two documents may be quite different even they are slightly different. This property makes SimHash an ideal technique for detecting near-duplicated ones, determining two documents are similar if their corresponding hash-values are close to each other. The closer they are, the more similar are these two documents; when the two hash-values are completely same, we actually find two exact duplicates, as what MD5 can achieve.

```
SimHash( document D )
{
01  Init vector Sim[0..(f-1)] = 0;
02  For (each feature F in document D) Do
03      F is hashed into an f-bit hash value X;
04      For (i = 0; i < f; i++) Do
05          If (X[i] == 1) Then
06              Sim[i] = Sim[i] + weight(F);
07          Else
08              Sim[i] = Sim[i] - weight(F);
09  For (i = 0; i < f; i++) Do
10      If (Sim[i] > 0) Then Sim[i] = 1;
11      Else Sim[i] = 0;
}
```

Fig. 2. Algorithm specification of SimHash.

In this project, we choose to construct a 64-bit fingerprint for each web document because it also works well as shown in [1]. Then the detection of near-duplicate documents becomes the search of hash values with $k-bit$ difference, which is also known as searching for nearest neighbors in hamming space [1,2]. How to realize this goal efficiently? One solution is to directly compare each pair of SimHash codes, and its complexity is $O(N^2)$, where N is the size of document repository and each unit comparison needs to compare 64 bits here. A more efficient method as proposed in [1] is implemented as well in this project. It is composed of two steps. Firstly, all $f-bit$ SimHash codes are divided into $(k+1)$ block(s), and those codes with one same block, say 1,2, …, $(k+1)$, are grouped into different list. For example, with $k=3$, all the SimHash codes with the same 1st, 2nd, 3rd, or 4th block are clustered together. Secondly, given one SimHash code, we can get its 1st block code easily and use it to retrieve a list of which all codes sharing the same 1st block as the given one. Normally, the length of such list is much smaller than the whole size of repository, N. Besides, given the found list, we need only check whether the remaining blocks of the codes differ with k or fewer bits. The same checking need to be applied to the other 3 lists before we find all SimHash codes, i.e. all near-duplicate documents. This

search procedure is referred as hamming distance measure by us in the remaining text.

B. Why query-biased and how it works

Although SimHash itself is effective in measuring how similar the given documents are, and the approach proposed in [1] to find SimHash codes with $k-bit$ difference can be efficient, the luxury needs of large cluster of machines and complex software systems prevent its wide acceptance.

In this project, our initiative to propose a more affordable, but still effective and efficient solution for detecting near-duplicate documents in a large repository comes from the following observations and thinking:

--Detecting duplicated documents among large repository requires intensive computing resources. Without advanced configuration, like high-performance server or cluster of machines, the processing time is easily become non-affordable and non-tolerable;

--Storage need is also quite large when we face a large document repository. To make the computation fast, we prefer to have the processing happened in RAM, but this becomes impossible easily when the number of documents to process and the k value increase. If we can't conquer this point, we won't have an efficient algorithm before turn to distributed computing environment;

--We notice two facts that, (1) the results retrieved given a specific query are much smaller in size compared with that of the whole document repository; (2) duplicated documents also exist in the returned results, therefore the total number of near-duplicate documents will become less if we could remove the duplicated ones from the retrieved results effectively;

--Divide-and-conquer strategy is possible here based on these two facts. A series of queries actually divide the target space into a set of smaller sub-spaces (Fig 3), and the processing on these smaller problems is expected to be much more affordable;

--Different k can be selected for different queries to achieve customizability as required in various applications. For example, if we want to ensure there as fewer as possible near duplicates appear in the results of popular (most frequent) queries, we can increase the value of k for those logged queries with highest frequencies. Reversely, smaller k may be enough for less popular queries, which will save computing resource. For those queries with very few results, $k=0$ is possibly welcome;

--The detecting and removing operation can be done in parallel. This query-based scheduling strategy allows very flexible task assignment, suitable for multi-threaded and/or distributed computing environment;

Actually, more merits can be found regarding to this method, given different target in different application environment, and this is left to interested readers.

From the discussion above, careful readers may have found that the overall idea of this method is "divide and conquer." The whole web space is divided into smaller clusters in which related documents are grouped given the queries (Fig 3). How this approach works is described as below, assuming a document repository is ready, $P=\{P_i\}\ (i=1..N)$.

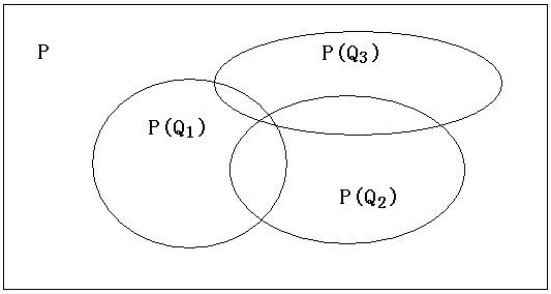

Fig. 3. An abstract view about the basic idea of this approach: divide and conquer. P refers to the whole document repository, and $P(Q_i)$ represents the set of documents returned given the query Q_i.

--Firstly, we rank those queries appearing in search log in a decreasing order in term of searching frequencies, denoted by $Q=\{Q_1,Q_2,...,Q_n\}$.

--Secondly, starting from the query with highest searching frequency, i.e. Q_i and $i=1$, we use it to search for related documents from the repository P, and the results are denoted as $P(Q_i)=\{P_1,P_2,...P_m\}_i$. These documents are ordered with someone ranking algorithm.

--Thirdly, we apply SimHash to encode the retrieved documents (see Fig 2) and grouping near-duplicate documents among $P(Q_i)$ based on Hamming distance measure, $P(Q_i)=\{\{P_j\}_1,\{P_j\}_2,...,\{P_j\}_k\}_i$ assuming there were k groups. Within each group $\{P_j\}_k$, the relative order of $\{P_j\}$ keeps constant as they are in the whole returned results.

--Then, some criteria is used to determine which documents in each group $\{P_j\}_k$ are to be deleted from $P(Q_i)$, so as in the repository P. One possible standard is discussed in the next section.

--If $i+1<n$, we continue with a new query Q_{i+1} in Q, and please note that the size of repository P is smaller than before due to the deletion of near-duplicate documents.

C. Work with PageRank

Near-duplicate documents can be found with the method discussed in the previous sections, but there is still one problem left, i.e. which ones to remove given a set of similar documents. In this project, we rank those similar documents in a decreasing order based on their authority scores determined by PageRank algorithm [5], considering it is a fair, reliable and, most importantly, query-independent measure about the authority of documents. In this project, we simply delete all documents other than the one with highest score. Applicants may define a threshold value in advance based on history experience, and remove those documents with scores smaller than that value.

This criterion can work with other rules to realize more complex application. For example, if the number of retrieved

documents about a query is too small, we may choose not to delete any duplicated documents. Some documents from specified sites may always be reserved for some business purpose.

By now, we have described why these algorithms and how they work. Whether they work well is demonstrated in the following section.

III. EXPERIMENTAL RESULTS

A. Overall description of experiments

We crawl and indexing about 1.6 million of documents, and use them as the testing repository. We refer to the search log recording a one-year long duration. Our study focuses on the effectiveness, efficiency and customizability of this query-biased near-duplicate detection algorithm.

For each query, we do the similarity checking among the top 1000 results for two reasons: (1) saving testing time, and (2) rare people care about the results following outside the first 100 pages, assuming 10 results are presented in one page and returned to searchers (as we find on modern commercial search engines).

In the experiments, we divide the top 2000 queries, as selected from our search log, into seven groups: [001-100], [101-200], [201-300], [301-400], [401-500], [501-1000] and [1001-2000]. For each query group, two kinds of data are collected: the number of deleted duplicates, and the time used in second. The same experiments are repeated for different k, say 3, 2, 1 and 0 in our studies, and we always begin with the same repository P for different k to ensure a comparable results.

The machine used for experiments is a general PC configured with Intel Due Core CPU of 1.6GHz and 2GB RAM. It hosts the document repository, the search engine application, and the testing program.

B. Effectiveness

Regarding to the effectiveness, we care about if the algorithm can really help us to find those near-duplicate documents, and how well they do.

Although the effectiveness of SimHash is mentioned in [1] and [2], we still ask several professionals to check the results manually at the beginning to confirm that the published findings and the correctness of our implementation, i.e. those deleted documents are really near duplicated. Table 1 records the basic statistics about the number of deleted duplicates and time used, for different k and different query group.

We observe that for the same k, fewer duplicates are deleted for less frequent queries. For example, when $k = 3$, 31097 documents are removed given queries [001-100], compared to 25996 documents by queries [101-200]. Fig 4 is about the average number of documents being removed per query in different group, when $k = 3$, and it also indicates this decreasing trend. Two causes can be contributed to explain this: (1) queries from [001-100] very likely have similar concept as some queries from [101-200], for example "TOYOTO" (assume it is from [001-100]) and "TOYOTO CROWN" (assume it is less frequent query and from [101-200]), considering that popularities are constant within a specific period and population; and (2) the processing of each previous query will result with fewer documents left in the repository. However, if the two adjacent queries represent two completely different concepts, like "TOYOTO" and "COFFEE", we may not observe this happening, or we say that they are not order sensitive.

When k increases, more duplicates are found and removed. As shown in Fig 5, with the top 2000 queries, totally 205607 duplicated documents are found and removed when $k = 0$, i.e. only completely same documents are removed as done with MD5; however, this number increases to 323823 when $k = 3$, about 50% more. This tells us that with non-zero k, both same and some similar documents can be found with SimHash technique, which ensures better usage experience than by MD5. This finding is consistent with that mentioned in [1,2].

TABLE I
THE NUMBER OF DUPLICATED DOCUMENTS DETECTED AND THE ELAPSED TIME USING QUERIES WITH DIFFERENT FREQUENCY
(TOTAL # OF DOCUMENT: 1612633)

k	Queries' range	# of deleted duplicates	Time used (s)
k=3	[001 – 100]	31,097	858
	[101 – 200]	25,996	693
	[201 – 300]	23,393	676
	[301 – 400]	23,359	682
	[401 – 500]	22,344	569
	[501 – 1000]	79,463	3,279
	[1001 – 2000]	118,171	5,536
k=2	[001 – 100]	28,009	911
	[101 – 200]	23,071	713
	[201 – 300]	20,672	677
	[301 – 400]	20,555	690
	[401 – 500]	19,520	577
	[501 – 1000]	70,784	3,336
	[1001 – 2000]	107,558	5,783
k=1	[001 – 100]	24,096	860
	[101 – 200]	19,465	681
	[201 – 300]	17,490	670
	[301 – 400]	17,211	674
	[401 – 500]	16,034	563
	[501 – 1000]	60,012	3,339
	[1001 – 2000]	90,255	5,919
k=0	[001 – 100]	20,393	1,098
	[101 – 200]	16,155	757
	[201 – 300]	14,486	689
	[301 – 400]	14,101	673
	[401 – 500]	13,044	574
	[501 – 1000]	50,706	3,495
	[1001 – 2000]	76,722	6,182

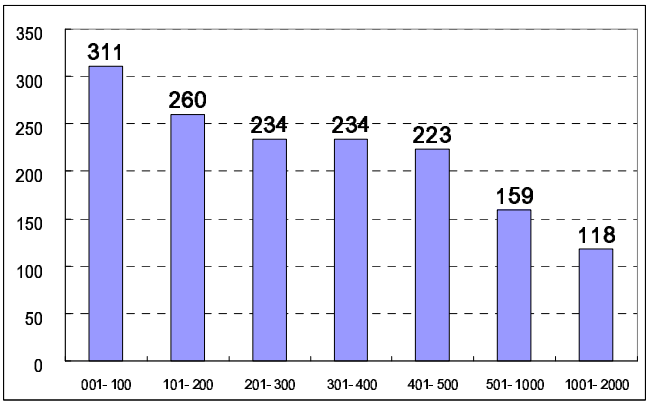

Fig. 4. The average number of duplicated documents being removed given queries with different frequencies, when $k = 3$, derived from Table 1.

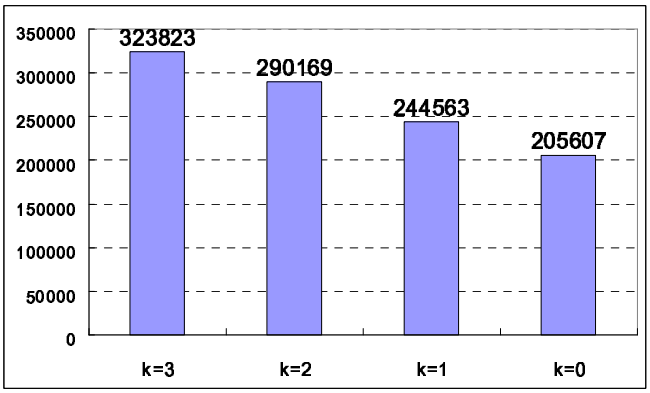

Fig. 5. The total number of duplicated documents being removed for different k, derived from Table 1.

C. Efficiency

We measure the efficiency in term of time in second, when k is set as 0, 1, 2 and 3 respectively (Fig 6), and two points can be concluded here:

1. Although more documents are detected and removed when $k = 3$ than $k = 0$, less running time is needed for $k = 3$. This is owned not only to the hamming distance measure technique used but to the promising feature of SimHash, similar documents have similar hash codes, so bit-operation burden is similar for different k;
2. About 6 seconds are required for each query's processing by average. This cost will increase when the size of repository increases. If we want to look for duplicates within more results, instead of 1000 in our experiment, we may have to afford longer waiting time. However, considering the effect discussed in the previous section, about 20% (= 323823/1612633) duplicates are found when $k = 3$ with top 2000 queries, this cost is acceptable.

Other than these two obvious points, there is one somewhat hidden but very promising advantage of this approach that may further increase the processing efficiency. Because each query can be processed separately and independently, we may configure more machines to do this job in a parallel manner. Those found duplicated documents are not deleted right now, but marked first. When all the parallel queries are done, the real deletion is applied to the repository finally then. This merit is expected to enable more efficient processing, although it is not implemented and verified in this project.

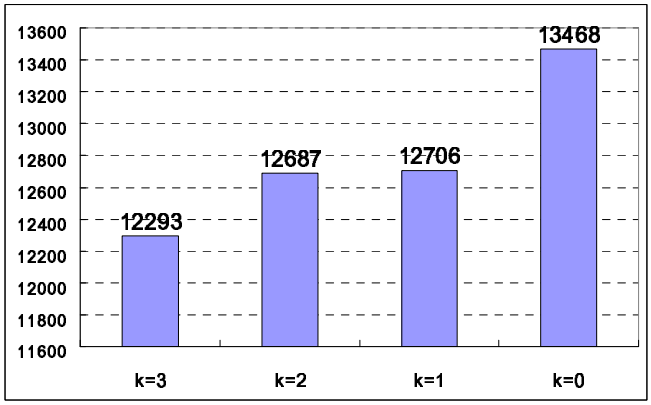

Fig. 6. The time used (in seconds) for different k, derived from Table 1.

This algorithm is preferred to run offline when, for example, new documents are added into the repository or the relative ranking of queries changes, so it won't slow down the search online.

D. Customizability

In addition to effectiveness and efficiency, flexibility is deemed as a very attractive aspect of this proposed framework:

1. Different k can be applied to queries with different frequencies. We have seen that increasing k will result in deleting more documents, so for those highly ranked queries, we may choose bigger k to ensure a high quality result. However, this may not be so necessary for those non-frequently used queries, where $k = 0$ or 1 would be enough;
2. Different k can also be applied to queries with different size of results. For those queries with very few results returned, we may prefer smaller k, say 0 or 1. This can work well with the following point;
3. Different checking size can be applied to different queries. In this project, we only search within the top 1000 results for duplicates, but this setting is free to adjust. For example, it may scale linearly with the number of documents returned.

E. Discussion

The designed experiments show us the promising potentials of this query-biased approach. With it, those exactly duplicated and near-duplicate documents among a large repository can be found with affordable time cost even on general PC. Besides, its inner nature makes the parallel processing possible, which ensures better performance if computing resource is not a problem. Finally, the related settings are customizable to satisfy different application requirements.

IV. CONCLUSION AND FUTURE WORK

In this paper, a query-biased near-duplicate detection algorithm is proposed. It demonstrates several promising

merits, like effectiveness, efficiency and customizability. Different from that in [1], it is more economical and affordable solution for wider population.

In this project, our choice of queries is based on their searching frequency as reflected in the search log. This is a reasonable decision, allowing us to spend more effort on those things with most attention, maximizing the ratio of investment and gain. However, we already notice that this may not be the optimizing choice because the top queries are not mutual, but may refer to the similar topic, e.g. "BASKETBALL GAME" and "NBA". Many of the results returned by these two queries may be same (Fig 3). A better choice would select queries about different interest, completely mutual (Fig 7) or with very small overlapping portion. Clustering technique, together with existing ranking of searching frequency, may be applied to make things better. Of course, some other rules can be figured out in different applications.

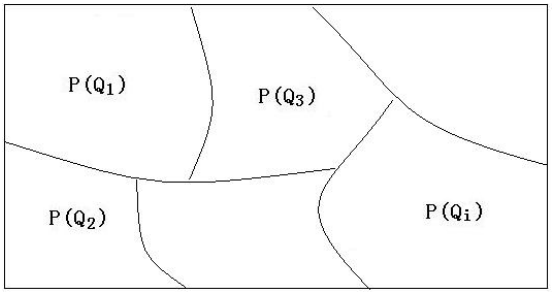

Fig. 7. An optimal division of the search space: all sub-spaces are mutual each other, or at most have tiny overlapping area.

REFERENCES

[1] G.S.Manku, A.Jain and A.D.Sarma, "Detecting near-duplicates for web crawling," *in Proc. 16th Int. World Wide Web Conference (WWW), 2007.*

[2] M.Charikar, "Similarity estimation techniques from rounding algorithms," *in Proc. 34th Annual Symposium on Theory of Computing (STOC)*, 2002, pp 380-388.

[3] A.Broder, S.C.Glassman, M.Manasse and G.Zweig, "Syntactic clustering of the web," *Computer Networks*, vol.29, no.8-13, 1997, pp 1157-1166.

[4] M.R.Henzinger, "Finding near-duplicate web documents: a large-scale evaluation of algorithms," *In Proc. ACM SIGIR*, 2006, PP 284-291.

[5] L.Document, S.Brin, R.Motwani, and T.Winograd, "The PageRank citation ranking: Bringing order to the web, " *Stanford Digital Libraries Working Paper*, 1998

[6] K.Bharat and A.Broder, "Mirror, mirror on the Web: A study of host pairs with replicated content, " *in Proc. 8th Int. World Wide Web (WWW), 1999.*

[7] S.Brin, J.Davis and H.Garcia-Molina, "Copy detection mechanisms for digital documents," *in Proc. ACM SIGMOD Annual Conference*, 1995

[8] http://en.wikipedia.org/wiki/MD5