Clustering and Load Balancing Optimization for Redundant Content Removal

Shanzhong Zhu
Ask.com

Alexandra Potapova
University of California
Santa Barbara

Maha Alabduljalil
University of California
Santa Barbara

Xin Liu
Amazon.com

Tao Yang
University of California
Santa Barbara

ABSTRACT

Removing redundant content is an important data processing operation in search engines and other web applications. An offline approach can be important for reducing the engine's cost, but it is challenging to scale such an approach for a large data set which is updated continuously. This paper discusses our experience in developing a scalable approach with parallel clustering that detects and removes near duplicates incrementally when processing billions of web pages. It presents a multidimensional mapping to balance the load among multiple machines. It further describes several approximation techniques to efficiently manage distributed duplicate groups with transitive relationship. The experimental results evaluate the efficiency and accuracy of the incremental clustering, assess the effectiveness of the multidimensional mapping, and demonstrate the impact on online cost reduction and search quality.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search Process, Clustering; H.3.4 [**Systems and Software**]: Distributed Systems

General Terms

Parallel Algorithms, Performance

Keywords

Parallel Near Duplicate Analysis, Incremental Computing, Scalability

1. INTRODUCTION

A large portion of pages crawled from the web have similar content. Detection of redundant content is an important data processing operation which facilitates the removal of near duplicates and mirrors, and also other low quality content such as spam and dead pages in search and web mining applications. Duplicate comparison algorithms have been studied extensively in previous work [5, 17, 6, 13, 8, 15, 21]. Since the precision requirement is high for removing redundant content in a production search engine, it is time consuming to use a combination of these comparison algorithms and conduct accurate computation for billions of documents in an offline system. As a web data collection is updated continuously, it is even challenging to keep an accurate and consistent view of near duplicate relations.

Major search engine companies have often taken a compromised approach: while a relatively small percentage of duplicates can be removed conservatively in the offline process, a large portion of duplicates are kept in a live online database. Duplicates are detected and removed during online query processing among those matched results. Such an approach simplifies offline processing, but increases the cost of online serving. From the cost saving perspective, it is attractive to reduce and balance online and offline expense because online expense is often proportional to the amount of traffic. Cost reduction is important for companies in developing cost-conscious search applications.

This paper discusses our experience in developing scalable offline techniques for removing a large amount of redundant content and provides evaluation results on their effectiveness. It addresses two technical issues arising in such a setting. First, we study the balancing of distributed workload on parallel machines by partitioning offline data. We use document lengths to guide data mapping which helps in the elimination of unnecessary machine communication for page comparisons. Because web document length distribution is highly skewed, we have developed a multidimensional mapping scheme to optimize load balancing and improve the overall data processing throughput during parallel comparisons. Second, because it is easier and less expensive to manage a cluster of duplicates instead of pair-wise duplicate pairs, the design of our scheme uses a clustering approach with incremental update and transitive approximation. We present our experimental results to demonstrate the effectiveness of our scheme and the impact on cost saving and search quality.

The rest of this paper is organized as follows. Section 2 discusses some background and related work. Section 3 discusses our overall design considerations. Section 4 presents the multi-dimensional mapping. Section 5 discusses incremental clustering. Section 6 presents experimental results. Finally, Section 7 concludes the paper.

WWW 2012, April 16-20, 2012, Lyon, France
ACM 978-1-4503-1230-1/12/04.

2. BACKGROUND AND RELATED WORK

Search services need to return the most relevant answers to a query and presenting identical or near-identical results leads to bad user experience. By removing duplicates, search result presentation would be more compact, and save users' time to browse results. Pair-wise comparison algorithms for duplicate detection have been studied extensively in previous works [5, 17, 6, 13, 8, 15, 21].

In general duplicate removal can be conducted in an online result displaying stage or in an offline processing stage. An offline data processing pipeline includes 1) a crawling system which continuously spiders pages from the web, 2) and an online data generation system that parses crawled pages and produces an inverted index for the online system to search. Duplicate detection and other data processing subsystems assist the main pipeline processing. The processing in the main pipeline is continuous as existing pages need to be refreshed to catch up with the updated content and new web pages need to be included as soon as possible.

Removing redundant content in an offline system requires a very high precision in duplicate judgment. Fast approximation algorithms [5, 17, 6, 8, 15, 21] are often not sufficient to simultaneously deliver high precision and high recall. A combination of these algorithms and increasing the scope of feature sampling with additional features can improve precision (e.g. [13]). This increases processing cost, especially for conducting accurate comparison with billions of pages. Accurate offline duplicate analysis with both high precision and recall can become a processing bottleneck and affect the freshness of online index. As a result, major search engines often resort to online duplicate elimination. In this online process, duplicates are detected among top results matching a query and only one result from duplicate pages is shown in the final search results. The disadvantage of delaying duplicate removal to the online process is that duplicates appear in the live index, which increases the serving cost. Several previous studies [10, 4] indicate that there are over 30% of near duplicates in crawled pages. Our experience shows that a significant engine cost saving can be achieved by removing these duplicates as we discuss in Section 6.

A related area in the database field is to study a similarity join operation which finds all pairs of objects whose similarity exceeds a given threshold. An algorithmic challenge is how to perform the similarity join in an efficient and scalable way. LSH [11] based approximation has been used to map pages with similar signatures into the same partition. Another study by Arasu et al. [2] shows that exact algorithms can still deliver performance competitive to LSH when using several candidate pair filtering methods including inverted index-based methods [20]. Additional filtering methods proposed are based on prefix and thresholds by Bayardo et al. [3] and Xiao et al. [23]. Other related work is sentence-level duplicate detection among web pages studied in Zhang et al. [24] with MapReduce [9]. MapReduce for parallel implementation is discussed in Lin [16] for top-k similarity computation and demonstrated by Wang et al. [22] for duplicate detection. The work by Peng and Dabek [19] presents systems support for incremental page processing using distributed transactions and notifications. Duplicate detection is accomplished through key lookup using content hash of pages or redirection targets. This technique is effective for handling exact duplicates, but is not targeted for similarity and near duplicate detection where complex all-to-all comparison is needed.

It should be mentioned that there are two types of duplicates among web pages: 1) *Content-based* duplicates where two web pages have near-identical content. This is measured by the percentage of content overlapping between the two pages. 2) *Redirection-based* duplicates when one web page redirects to another, either through permanent or temporary HTTP redirect. Although the source and destination of redirection have identical contents, the crawler may visit them at different times and download different content versions. For two pages q_1 and q_2, they are considered duplicates or near-duplicates if a pair-wise duplicate detection function $F(q_1, q_2)$ is true or if two pages redirect to each other through a sequence of redirection. For content-based duplicates, the similarity function between two pages q_1 and q_2 is defined as $Sim(q_1, q_2)$. Let τ be the control threshold of near-duplicate similarity, $F(q_1, q_2)$ is true if $Sim(q_1, q_2) > \tau$. The primary metric for computing $Sim(q_1, q_2)$ in the previous work [5, 17, 6, 13, 21] has been the Jaccard formula while the Cosine formula can also be a choice [12, 15].

3. DESIGN CONSIDERATIONS

Our task is to shift the most of duplicate removal operations from the online engine to offline processing. The key saving is that the online service does not need to host an index with duplicated documents and an online engine can collect more candidate results without duplicates in matching a query when there is a limitation on the total number of candidates selected. In this way, an online mult-tier serving architecture can become more effective. We discuss evaluation results in Section 6.

For offline near duplicate removal from a large dataset, we first study parallelism optimization for task and data mapping to improve load balancing and reduce communication. We assume exact duplicates are removed first in the offline data processing using techniques such as representing each page using one hash signature [19].

Our objective is to add an offline service with the following functionality: given a set of newly crawled pages, this service answers if a page is a duplicate of others or not. The result of such a query is advisory in the sense that a page may be indexed and delivered to the live online database before its duplicate status is determined and in that case, the system can decide if such a page should be removed from the online database or not. In this way, the duplicate analysis does not become a bottleneck for the main pipeline in delivering fresh documents to the online database. On the other hand, this duplicate analysis service needs to have a high throughput so that the offline system can notify the duplicate status of a page as soon as possible.

The focus of our optimization to improve the overall throughput of an offline parallel near duplicate service is in two aspects: 1) data partitioning for load balancing; 2) distributed duplicate information management and incremental update. We will discuss these two issues in Sections 4 and 5.

4. LOAD BALANCING WITH MULTIDIMENSIONAL PARTITIONING

The duplicate service uses a pair-wise algorithm for identifying duplicate pairs. Since many web pages do not change for a long period of time, we only compare a set of new or

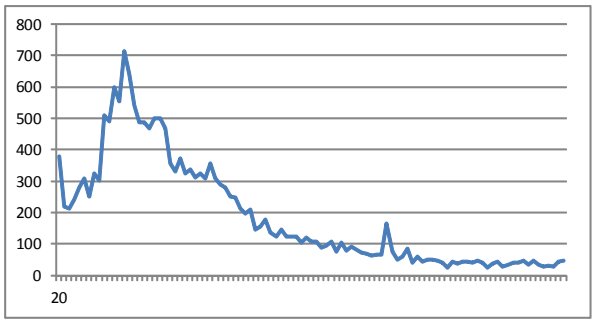

Figure 1: Size distribution for a web page collection. The X-axis lists the page size ranged from 20 to 1686 words. The Y-axis is the number of pages with each particular size.

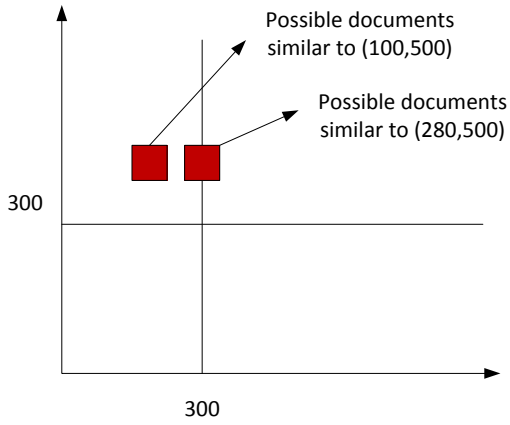

Figure 2: Examples of mapping a page with a 2D length vector to partitions.

updated pages with existing pages. In using a cluster of machines for a many-to-all comparison service, we need to map data and computation to each machine.

We choose to use page based mapping which assigns each page to one partition and assign one or few partitions to a physical machine. Another option for machine mapping is to use a fingerprint hash value based on shingling[4] or LSH [11]. There are two reasons we did not use the later mapping. First, such a method maps a page to multiple values and similarity computation counts the match over multiple fingerprint computing functions for each page pair, which increases inter-machine communications for result aggregation. Such overhead is significant considering the number of near duplicate pairs is huge. Second, to deliver a high accuracy and recall in near duplicate detection, a combination of multiple methods with different types of features [13, 12] is needed in practice. For example, Henzinger [13] reported 38% precision for the shingling algorithm by Broder et al. [5] and 50% precision for Simhash by Charikar [6] and a combined algorithm achieves 79% precision. Ask.com has combined multiple pairwise comparison heuristics with various features to reach high precision and high recall for offline duplicate removal. Thus, it is not easy to map documents to machines using a shingling or LSH based mapping.

We have the following design objectives for data and computation mapping. 1) Pages assigned to different machines should be less likely to be near duplicates so that we send a new or updated page to least numbers of machines for comparisons. 2) The number of pages assigned to each machine should be about the same. This helps load balancing of comparison computation because load generated on each machine is approximately proportional to the number of pages assigned. 3) As page content changes, page-to-machine mapping may need to be changed. The algorithm should minimize the chance of remapping as this causes an overhead in data migration and service disruption.

We consider to use the page length to guide the mapping of data to machines. For similarity join, Arasu et al. [2] describes the use of feature set size for dividing a general similarity join instance by observing that two sets have similar sizes if they have high Jaccard similarity. For near duplicate detection on a single machine, Theobald et al. [21] described a condition using the length of signatures to identify pages that cannot be duplicates. Motivated by such approaches, we initially have used the page length to map documents. Our experiments find that one dimensional data mapping based on the length of pages does not balance the computations well across machines as web document length distribution is highly skewed. Also it is not easy for machine load balancing to adapt to page length change. Figure 1 shows the sampled size distribution of 2 million pages randomly selected from a 100 million page dataset. The X-axis is the size value from 20 to 1686 words, and the Y-axis is the number of pages in the corresponding page size. The size distribution has certain spikes and high concentration at sizes around 237. With a skewed distribution of web page lengths, there are some partitions containing an excessive amount of pages, which becomes a throughput bottleneck. Often, one heavily-loaded partition has much more pages, which significantly slows down the overall throughput of a many-to-all comparison service.

4.1 A multidimensional mapping algorithm

We have developed a multi-dimensional algorithm that maps data to machines with better load balancing while assigning dissimilar pages to different partitions as much as possible. For simplicity of presentation, we first assume each machine hosts one partition. When there are larger numbers of partitions, we can evenly map partitions to machines.

This algorithm uses a combination of language and multi-dimensional signatures to partition web pages. It first partitions pages with respect to their languages (this paper assumes to deal with English documents only). It segments an English dictionary into N disjoint subsets, and then splits each page document into N subdocuments. Each subdocument only uses words in the corresponding dictionary subset and has a feature length computed only based on these words. Therefore, we can obtain N length features for each page, one for each dimension. Next, we describe how to partition each dimension.

- We slice and divide values at each dimension i into d_i intervals. The slicing method is discussed in the next subsection. We define a length range of N dimensions as a tuple containing a specific interval for each dimension. The total number of N-dimensional ranges is $d_1 \times d_2 \cdots \times d_N$. Each partition contains pages

whose length vectors within a specific N-dimensional range. For example, in a 2-dimensional setting illustrated in Figure 2, assume that each dimension is divided into two intervals $[0, 300)$ and $[300, \infty]$. There are 4 ranges: $([0, 300), [0, 300))$, $([0, 300), [300, \infty))$, $([300, \infty), [0, 300))$, and $([300, \infty), [300, \infty))$. Thus, we can create 4 corresponding partitions and every page will be mapped to one of them.

- When receiving a new or updated page, an N-dimension length vector for this page (called A) is computed, and this page will be further sent to machines that host partitions with pages possibly similar to A. We would like to send this page to the least possible number of partitions for comparison. For the example in Fig. 2, when $A = (100, 500)$ with threshold $\tau = 0.9$, only one partition containing length range $([0, 300), [300, \infty))$ is contacted for further duplicate comparison. If $A = (280, 500)$, then two partitions with ranges $([0, 300), [300, \infty))$, $([300, \infty), [300, \infty))$ will be contacted.

  Formally, let A^k denote a set of k-gram shingle signatures derived from a page feature vector A. let A_i^1 denote the subdocument in the i-th dimension of A^1, and let $L(A_i^1)$ be the length of subdocument A_i^1. Notice that this length function is the summation of frequencies of all words appearing in this subdocument. Let vector $(L(A_1^1), L(A_2^1), \cdots, L(A_N^1))$ be an N-dimensional length vector of this document. We can show that only documents whose length vectors fall into a bounding space with the following i-th dimension range might be candidates of being near duplicates to A:

$$(\frac{L(A_i^1)\tau}{\rho}, \quad \frac{L(A_i^1)\rho}{\tau}) \tag{1}$$

  where ρ is an interval enlarging factor defined in the next subsection.

  Thus feature vector A only needs to be sent to machines that host partitions with ranges intersecting with the above bounding space. Namely only those partitions can contain pages which may be near duplicates of A and a justification is provided below.

4.2 Similarity analysis

We analyze partitions that should be compared when identifying near duplicates of a page with feature vector A. We use the Jaccard similarity of two pages represented by consecutive k-gram shingles as defined by Broder et al. [5, 4]. Let $|A^k|$ be the set size which only counts its unique members. In computing k-gram shingles of a page, we append $k-1$ dummy words at the end of this document. When k is not too small, our experiments show that $L(A^1)$ is relatively close to $|A^k|$ and let ϵ be the relative difference ratio between these two terms satisfying $1 - \epsilon \leq \frac{|A^k|}{L(A^1)} \leq 1 + \epsilon$.

There is a small length variation δ in distributing words in page A into N subdocuments. Namely

$$|\frac{L(A^1)}{N * L(A_i^1)} - 1| \leq \delta.$$

We define the interval enlarging factor $\rho = \frac{(1+\delta)(1+\epsilon)}{(1-\delta)(1-\epsilon)}$.

For another existing page $B = (B_1, B_2, \cdots, B_i)$ which does not fall in the range defined in Expression (1), there exists dimension i such that

$$L(B_i^1) \leq \frac{L(A_i^1)\tau}{\rho}, \text{ or } L(B_i^1) \geq \frac{L(A_i^1)\rho}{\tau}. \tag{2}$$

Note that the Jaccard similarity of these two pages with respect to k-grams is

$$Sim(A, B) = \frac{|A^k \cap B^k|}{|A^k \cup B^k|}.$$

Given the above terminology, we show that A and B cannot be duplicate. Since $\frac{|A^k|}{L(A^1)}$ and $\frac{|B^k|}{L(B^1)}$ are within $[1-\epsilon, 1+\epsilon]$, and

$$\frac{|A^k \cap B^k|}{|A^k \cup B^k|} \leq \min(\frac{|A^k|}{|B^k|}, \frac{|B^k|}{|A^k|}),$$

we have

$$Sim(A, B) \leq \min(\frac{|A^k|}{|B^k|}, \frac{|B^k|}{|A^k|}) \leq \min(\frac{L(A^1)}{L(B^1)}, \frac{L(B^1)}{L(A^1)})\frac{1+\epsilon}{1-\epsilon}.$$

Thus, we can work on a multidimensional condition that operates on unigrams. In our scheme, A is divided into N subdocuments under N sub-dictionaries and each of them produces a unigram signature set $A_1^1, A_2^1, \cdots, A_N^1$ respectively. The distribution satisfies the following inequalities

$$(1-\delta)N * L(A_i^1) \leq L(A^1) \leq (1+\delta)N * L(A_i^1)$$

and

$$(1-\delta)N * L(B_i^1) \leq L(B^1) \leq (1+\delta)N * L(B_i^1).$$

From the above two inequalities, we have

$$\min(\frac{L(A^1)}{L(B^1)}, \frac{L(B^1)}{L(A^1)}) \leq \min(\frac{L(A_i^1)}{L(B_i^1)}, \frac{L(B_i^1)}{L(A_i^1)})\frac{1+\delta}{1-\delta}.$$

Based on condition (2),

$$\min(\frac{L(A_i^1)}{L(B_i^1)}, \frac{L(B_i^1)}{L(A_i^1)}) \leq \frac{\tau}{\rho}.$$

Thus,

$$\min(\frac{L(A^1)}{L(B^1)}, \frac{L(B^1)}{L(A^1)}) \leq \tau\frac{1-\epsilon}{1+\epsilon}.$$

Hence the Jaccard similarity of these two pages with respect to k-grams satisfies $Sim(A, B) \leq \tau$. Namely A and B cannot be a near duplicate pair.

The interval enlarging factor ρ is estimated from computation of δ and ϵ, which are sampled in our implementation with an approximation. Since the above analysis between vectors A and B uses worst-case bounds in inequality derivation, there is a room to tighten ρ while still preserving the correctness approximately. We have tested a ρ value between 1 and 1.3 and the above approximation may cause some partitions not to receive pages to compare and this can reduce duplicate recall. We evaluate its impact on accuracy in Section 6.

4.3 Dimension slicing and partition mapping

We discuss how each dimension is sliced. For a 1D size space, we first consider SpotSigs' partitioning strategy [21] which scans size values of all pages from smallest to biggest and merge consecutive values to produce d intervals with the

left bounds as $p_1, p_2, \cdots p_d$ and those values satisfy $\frac{p_i}{p_{i+1}} < \tau$. For example, given a size list: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 with $\tau = 0.8$, the following intervals are produced: [1,2), [2,3), [3,4),[4,6), [6,8), [8,10]. We call these intervals fine-grain intervals.

Next we consider page distribution under the above fine-grain interval and merge them into coarse-grain intervals so that the number of pages in each coarse-grain interval is about the same. Given k fine-grain intervals, we consecutively consolidate some of these intervals with their neighbors and map them to targeted S intervals so that each interval has close to M/S pages where M is the total of documents to be mapped. For example, given a page distribution 2, 3, 4, 5, 2, and 1 for the following fine-grain intervals [1,2), [2,3), [3,4),[4,6), [6,8), [8,10], respectively. To produce 4 balanced intervals, we have [1,3), [3,4),[4,6), [6,10] so that 5, 4, 5, and 3 pages are distributed into each interval respectively. If the original page distribution is 2, 3, 4, 10, 2, and 1 for six fine-grain intervals, then the merged intervals have the following distribution 5, 4, 10, and 3. This example also illustrates that it is not easy to balance a skewed distribution using a 1D mapping.

For an N size space, we apply the above 1D slicing and merging method at each dimension to balance page distribution at each interval. Multidimensional mapping is more effective in dealing with distribution spikes compared to 1D as the 1D page size is approximately divided by N at each dimension. When the data set is updated and expanded, the multi-dimensional mapping is less sensitive to size variation.

5. DUPLICATE CLUSTERING AND INCREMENTAL UPDATE

Once duplicate pairs are detected, the offline system keeps the winners of duplicates in the online database and remove the losers. Winners are pages that need to be kept in the final database while losers are pages that need to be removed. There are a number of page features that can be used for winner selection. For example, pages with a high link popularity or frequently clicked in the search logs may be selected. Such features are used in the query result ranking [18, 14, 7, 1]. When the popularity score of pages is about the same, URLs with a long length may be less preferred compared to a shorter one and a dynamic page may be less preferred compared to a static URL. When pages are selected for a different market, there could be additional rules imposed. For example, a database produced for the United Kingdom may prefer hosts ended with .uk domain instead of .com or .org.

It is not easy to apply the above rules to duplicate pairs. There are two reasons. 1) We often cannot get a consistent or stable winner selection among duplicate pairs. Some pages are initially considered as losers and can become winners later on. For example, with two pairs of duplicates (x, z), (x, y), if page x is the winner for the first pair, and it is a loser for the second pair, then both x and y will be kept. 2) When a winner becomes dead, we would need to revisit its losers quickly and find a replacement for this winner from these losers. Storing all duplicate pairs explicitly is expensive as there are a huge number of duplicate pairs.

Given the above considerations, we choose to cluster duplicates in a transitive relationship, that simplifies duplicate data management. Each page either does not have any duplicate or belongs to one and only one duplicate group. Two duplicate groups are merged if there exists a duplicate pair between one page from the first group and another page from the second group. When answering a query on the duplicate status of a page, if this page is a loser in a group, the system verifies its similarity threshold with the group winner to ensure detection precision.

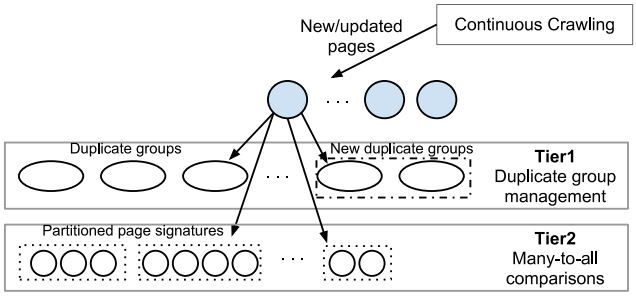

Figure 3: Two-tier architecture for incremental duplicate clustering with continuous data update.

This has three advantages. 1) Winner selection is always conducted in a group of pages, which yields a relatively stable selection compared to a pair-wise manner. Once duplicate groups are detected, a database generation scheme can conduct winner selection and use the selected winners to produce different online databases for different search markets. 2) Space cost for storing such duplicate groups has an affordable linear complexity, proportional to the number of pages. 3) Clustering also allows a simpler integration with incremental computing. If content of a page is changed, the previously computed duplicate status regarding this page may not be valid any more. We can revise the previously detected groups incrementally.

5.1 Distributed Group Management and Information Serving

We discuss how duplicate groups are constructed and updated as pages are added or updated to an offline data collection. Given a set of pages recently updated or added, we need to identify if they belong to existing duplicate groups, or form new duplicate groups. As shown in Figure 3, we use a two-tier scheme to detect and maintain duplicate group membership as data is being updated.

- Tier 1 service is to manage duplicate groups detected previously from all pages in the database and use such information without performing extensive comparisons to answer if a page is a duplicate of others or not. There are a large number of duplicate groups to be managed in Tier 1. We can distribute those groups to a large number of machines so that each machine is responsible for a subset of duplicate groups. Such a cluster can quickly answer if a page belongs to an existing duplicate group or not.

  Tier 1 service examines if given page q_1 has an existing duplicate group. If q_1's content has not been changed much, then the system can make a decision of whether this page should still stay within this group without going through a full comparison with other pages. To facilitate this, we use a group representative signature vector R to model the content characteristic of this

group. An updated page q_1 stays within this group if $F(q_1, R)$ is true.

This representative signature is normally the signature of the winner page in the group because such a page tends to be more stable to stay within the group. When a winner page has a content change, we will update the representative signature. We don't use feature aggregation such as a centroid of all signature vectors to represent a group. That is because it is more meaningful to assess the impact over the relevancy when using this winner to represent other losers in the online search results. Since this winner will be in the live index. Besides, it also allows us to impose a strict rule to double check the winner's signature with a loser and reduce error caused by transitive clustering.

When there is no existing duplicate groups found for a given page q_1, or page content of q_1 has been changed significantly and it does not belong to its current group, then this page needs to be compared with other pages. We defer this operation to tier 2 detection service.

- Tier 2 conducts a sequence of many-to-all comparisons or all-to-all comparisons if needed to derive duplicate information that Tier 1 does not have. This process repeats while more data is being sent to the duplicate service for processing. It accumulates a set of new or updated pages sent from Tier 1 when tier 1 cannot make a decision which group pages should belong to as discussed above. If a duplicate of a page q_1 is found through Tier 2 service, then Tier 1 updates its duplicate groups by letting q_1 join the newly-detected group. If no duplicates of q_1 are found, q_1 forms a new group which only contains one member. This tier can use the multi-dimensional partitioning and mapping algorithm discussed in Section 4. The reason that Tier 2 accumulates a set of pages before starting the comparison is to reduce the excessive overhead such as I/O in loading signatures of other pages from the secondary storage to memory.

When pages are updated, duplicate groups among them need be updated too. We describe group splitting and merging rules to maintain content-based and redirection-based duplicates as follows. Given a page q_1 crawled and added to the database at time t, there are three cases to handle.

- Case 1: q_1 is an existing page with the updated content and page q_1 does not redirect to another page. Let current duplicate group of q_1 be g_1, detected before time t.

  If q_1 is found to be a duplicate of another page q_2 in group g_2 through Tier 2 service after time t, then q_1 needs to be removed from g_1 and added to group g_2. Other pages that redirects to q_1 in the current group g_1 should also be moved to g_2.

  If q_1 is not a duplicate of any existing page, then q_1 forms a new group along with any other known pages that redirect to q_1. For those pages in g_1 that redirect to q_1, they need to be moved out and join this new group.

- Case 2: page q_1 does not redirect to another page, and q_1 is a new page.

  If q_1 is found to be a duplicate of an existing page q_2 after time t through Tier 2 service, then q_1 is added to the current group of q_2.

  If q_1 is not a duplicate of any existing page, then q_1 forms a new singleton group.

- Case 3: page q_1 redirects to another page q_t directly or indirectly through multiple levels. $q_1 \rightarrow q_2 \rightarrow \cdots \rightarrow q_t$ where q_t has the real content. We apply the same rules as described in Case 1 and Case 2 for q_t to identify the group that q_t should join. In addition to this, any page that redirects to page q_t (directly or indirectly) need to be moved to the new group of q_t.

5.2 Tradeoff of Performance with Accuracy

Clustering and distributed duplicate data management improves performance, but plays a tradeoff for accuracy. We discuss the impact of these techniques on accuracy and additional techniques used to reduce error. Two types of error are discussed. A "relative error in precision" occurs in cases when a pair of pages is classified as duplicate by our scheme while the baseline disagrees. A "relative error in recall" occurs in cases when a duplicate pair identified by the baseline algorithm is not detected in this two-tier scheme.

- **Transitive clustering.** Pages clustered by transitive relationship carries an approximation that affects detection precision while it could improve the duplicate recall.

- **Distributed duplicate group management**.

  - **Group splitting.** When a duplicate group splits according to the above design, the algorithm does not recompute the duplicate relationship within each divided group and sometime it may create a false positive. Specifically, when $F(q_1, q_2)$ is true and $F(q_2, q_3)$ is true before time t, all of these three pages are in one group. If content of q_2 changes after t and it is moved out from the group, pages q_1 and q_3 still stay as one group after time t even there does not exist any page in this group to connect q_1 and q_3 transitively.
  - **Similarity with group representatives.** We compute the similarity with group representatives to reconfirm the group membership of a page with small content change. That speeds up processing, but can affect precision and recall. We discuss two cases as follows. 1) There are two pages q_1 and q_3 such that $F(q_1, q_2)$ and $F(q_2, q_3)$ are true before time t. After time t, the new version of q_2 is similar to the group representative, but there is no page to bridge the duplicate relation of q_1 and q_3 transitively. 2) Also since the new version of page q_2 is found to be close to the group representative signature in Tier 1, Tier 2 comparison is skipped. However, there can exist pages stored in Tier 2 machines which are near duplicate to this new version of q_2.

- **Content update latency**. The detection scheme relies on page features to conduct comparison. Given a distributed architecture, there is a latency for receiving the latest version while the comparison tasks are

conducted concurrently and continuously. Thus the staleness of page content due to update latency causes a certain degree of precision and recall loss.

The loss of duplicate recall only affects the scope of cost reduction in the online engine because online duplicate removal is still available. It is not critical for our goal as long as the percentage of recall loss is small. The impact of precision loss can be significant to engine's relevancy.

We have used two strategies to retain detection precision. 1) This duplication processing service is mainly used when the offline indexing pipeline asks for the duplicate status update of pages after they are recrawled or discovered. Thus the system verifies again the similarity of a queried page with its group winner. If the similarity falls below a threshold, we will not report this page to be a duplicate of this winner. 2) When there is a significant number of reports indicating member's dis-similarity with the winner within a group, this group needs to be repaired. One way is to recompute the duplicate membership of this group. With these strategies, our experience is that the above verification process rejects less than 8% of clustered near duplicate pairs and the cost of repairing duplicate groups is small in terms of frequency and time. Thus these clustered near duplicates which satisfy the required similarity threshold compared to winners can be removed in offline.

6. EVALUATIONS

Our evaluation has the following objectives. 1) Demonstrate the usefulness of incremental computation and the effectiveness of using representative group signatures in saving computation cost. 2) Examine the impact of load balancing and compare 1D, 2D, and 3D mappings for parallel comparisons, and demonstrate the speedup of the service throughput as the number of machines increases. 3) Assess the accuracy of the two-tier architecture, the overall impact of aggressive offline duplicate elimination on the cost and relevancy of the online search engine.

We have implemented an incremental offline duplicate handling system with C++ at Ask.com's platform and used it for processing billions of web pages. This system uses a pairwise signature comparison algorithm developed internally at Ask.com with multiple heuristics and a high threshold to achieve high precision and recall rates. Because of the high precision and recall requirement for offline duplicate removal, an earlier implementation of all-to-all comparison based on this pairwise algorithm is time consuming and has become a bottleneck for the data processing pipeline. The use of this incremental solution with the optimized mapping and balancing scheme speeds up the overall data processing by one order of magnitude. Clustered duplicate groups are also used for various purposes including page selection for online database for different markets, bad/spam page removal, and crawling control. We discuss our experience and evaluation results in this section. For the demonstration purpose, we have also used a subset of a data collection crawled in three weeks with a size grown from 10 million URLs to 100 million URLs. Some of those pages were recrawled several times during this period. The experiments are conducted in a cluster of machines with dual-core Xeon 3GHz and 6G memory.

6.1 Effectiveness of multi-dimensional mapping

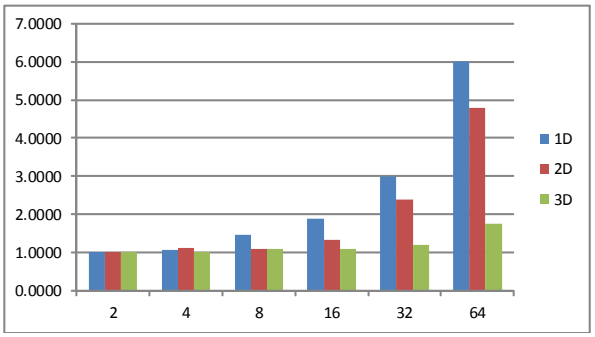

Figure 4: Load imbalance factor of 1D, 2D, and 3D mapping when the number of machines varies from 2 to 64.

We assess the effectiveness of load balancing with 3D mapping as a dataset of 100 million pages is being updated. The load imbalance factor is computed as the ratio of the maximum page numbers hosted in a machine over the average number among all machines. Figure 4 shows load imbalance factor of 1D, 2D and 3D mapping on the Y-axis with a different number of machines on the X-axis. The reason for 1D mapping has a higher load imbalance is that the skewed size distribution caused a difficulty in assigning pages evenly to partitions while satisfying the minimum similarity overlapping condition. Both 2D and 3D mapping can improve load balancing while 3D mapping gives more flexibility and performance gains. As the number of desired partitions (or machines) increases, the imbalance factor for 3D becomes relatively smaller than others. Increasing the dimension number further does not offer significantly more benefits and because of this reason, we have chosen 3D mapping for the production setting.

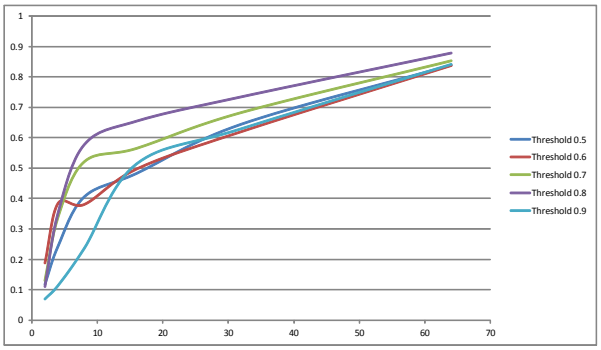

Figure 5: Percentage of inter-machine communication reduced with 3D mapping.

Figure 5 shows communication saved with 3D mapping and mapping. The X-axis is the number of machines and the Y-axis is the percentage of reduction in terms of the number of web pages communicated using the 3D algorithm in Section 4. To estimate communication saving, we compute the total number of web pages that need to be exchanged among machines and the total number of pages that need not to be sent. With more machines, the communication volume among machines increases for exchanging pages. 3D

mapping considers partition dissimilarity and can still effectively avoid unnecessary communication.

Figure 6 shows the throughput speedup ratio of a single machine over a cluster setting with 2, 4, 8, 16, 32, 64 machines in processing parallel comparisons. In the single machine setting, we record the maximum number of new or updated pages it can handle per second for Tier 2 duplicate comparisons. For each cluster setting, we record the overall maximum number of new or updated pages it can handle per second. The 3D scheme delivers much more speedup than 1D scheme when the number of machines increases and the 1D scheme faces more challenges in balancing workload and thus overloaded machines slowdown the entire system throughput significantly. For the case of 64 machines, the throughput with 3D mapping is 3.3 times faster than 1D.

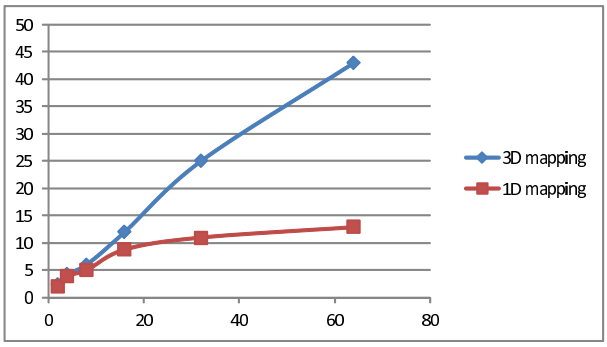

Figure 6: Speedup of processing throughput when the number of tier 2 machines increases from 2 to 64 for parallel comparisons.

3D mapping also has an advantage in adapting to page size variation as the database content changes. Because mapping of partitions to machines is based on the sampled data, we compute the coefficient of variation (CV) of load imbalance factors when datasets change. As an example, we use ten different datasets after initial mapping is determined through a training dataset. CV is defined as the standard deviation divided by the mean value of load imbalance factors. The CV value is 4.5% for 1D mapping while it is 0.29% for 3D mapping. Thus in addition to having a better load balance, 3D mapping can be less sensitive to size variation when content of a data collection changes.

6.2 Effectiveness of incremental computation

Figure 7 shows the processing time ratio of our incremental detection method over a non-incremental detection approach in a single server when a data collection hosted increases from 0 to about 100 million URLs on 50 machines. X axis is the time when the duplicate detection time is recorded in comparing two approaches. Initially, the machine has a very small data collection, the per-page response time in detecting its duplicates non-incrementally is smaller than the one with an incremental approach. Then the ratio in Y axis is less than 1. As the database increases, it takes a longer time to conduct comparison from scratch and thus per-page response time using a non-incremental method is slower than the incremental version. Thus the ratio in Y axis starts to exceed 1 and reach a 24-fold improvement at the end.

Figure 8 illustrates the usefulness of incorporating representative group signatures in tier-1 groups, which avoids

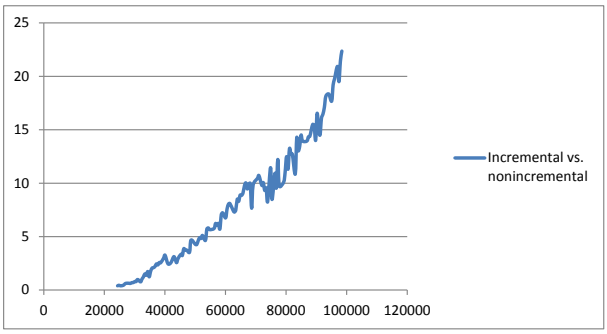

Figure 7: Ratio of non-incremental duplicate detection time over incremental detection time when the data collection size increases from 0 to 100 million URLs.

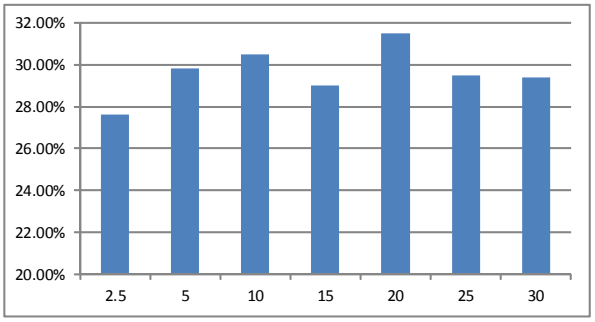

Figure 8: Percentages of updated pages with signatures similar to representative group signatures.

certain comparison cost. In this experiment, we start with 30 million URLs crawled in one week, and then examine another 30 million URLs crawled one week later. There are about 50% of URLs overlapping between these two sets of URLs. In processing the second set of URLs, we record the percentage of updated URLs that have signatures similar to representative group signatures within the defined threshold as near duplicates. Figure 8 shows the percentage of updated URLs similar to group signatures when the number of pages processed in the second batch is 2.5, 5, 10, 15, 20, 25, and 30 millions, respectively. The result shows that around 30% of updated URLs do not have a significant content change, and are still similar to group signatures. The computation for those URLs is localized within tier 1 and comparisons with tier 2 datasets are skipped to save cost and improve the tier-2 service throughput.

6.3 Accuracy of distributed duplicate clustering and group management

Next we compare detection error of our offline analysis service distributed on a cluster of machines with a single node non-parallel setting where the proposed approximation techniques are not applied. In this experiment, our data collection was crawled in three weeks with a size grown from 10 million URLs to 100 million URLs. Some of those pages were recrawled several times during this period. To establish a baseline performance, we run a centralized comparison among all 100 million pages to establish duplicate groups using the Ask.com's pairwise comparison procedure.

Then we replay the trace of crawling to send these 100 million pages gradually to a two-tier system. When the data collection size reaches each milestone in 10 millions, 20 millions, 30 millions, 40 millions, 75 millions and 100 millions, we measure the relative error in precision and recall compared to the single-machine solution. Table 1 show relative error rates in percentages when the dataset grew from 10 millions, 20 millions, and up to 100 millions. Three configurations are tested with 12, 24, and 36 machines in Tier 2 respectively. The result shows that the accuracy impact of the approximation techniques in the distributed design is fairly small.

Table 1: Percentage of relative error in precision (REP) and in recall (RER) in duplicate clustering. The number of partitions (machines) is 12, 24, and 36.

	10M	20M	30M	50M	75M	100M
REP, 12	0.8%	0.85%	0.9%	0.85%	1%	1.3%
RER, 12	1.5%	1.55%	1.6%	1.67%	1.6%	1.65%
REP, 24	1.1%	1.2%	1.22%	1.29%	1.3%	1.25%
RER, 24	1.6%	1.64%	1.59%	1.68%	1.7%	1.74%
REP, 36	1.2%	1.07%	1.09%	1.1%	1.1%	1.1%
RER, 36	1.7%	1.7%	1.65%	1.69%	1.7%	1.7%

6.4 Impacts on relevancy and cost

While the use of this offline duplicate removal solution has increased the overall index processing speed by an order of magnitude, this scheme does not affect the freshness of the search engine index. The freshness is measured in terms of the latency from the time a URL is discovered to the time this URL appears in the live index if it is not a duplicate. This is because the offline pipeline queries the duplicate statues of newly crawled and updated pages, and continues to send URLs to the live search engine without waiting for a response from the duplicate analysis service. The pipeline uses the previously developed results stored in the pipeline for online database update. Once the new duplicate status is received for a set of pages, the pipeline updates the stored duplicate information.

A number of tests have been conducted to evaluate the accuracy of offline page removal and the relevancy impact to the queries. One experiment was to collect top 10 results from other search engines such as Google for 400K queries in an extended period. These top results are considered to be non-duplicate to each other. To assess if our duplicate handling system removed some of them by mistake, those top URLs from other engines which are classified as a loser in our system were identified and their corresponding winners selected by our system were checked to see if duplicate relation is true or not. Dividing the number of incorrect loser-winner pairs found by the total number of URLs examined gives the error rate of duplicate judgment. The monitoring result showed that the daily error rate of duplicate judgment was from 0.017% to 0.028%. Many of the failed cases have Javascript/flash content which is difficult to handle accurately.

We assess the cost saving by shifting most of near duplicate removal from online to offline. We compare the following two schemes after the exact duplicates are removed first:

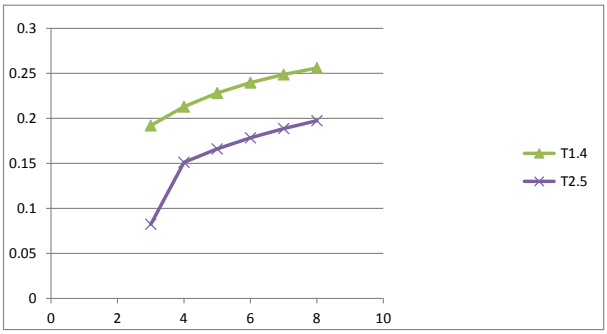

Figure 9: The saving ratio of total machine cost with a fixed QPS target when the database size changes.

- A) The offline system first uses a conservative scheme to remove about 5% of all pages and the online scheme removes additional duplicates among the top results in answering a query.
- B) The offline system with our approach removes near duplicates as much as possible (about 33% of all pages) and the online scheme removes the remaining duplicates on the fly.

Figure 9 shows the percentage of total machine reduction by scheme B compared to A in accomplishing the same online traffic capacity, which is measured by the number of user queries that can be handled per second. The x-axis is the database size processed in the offline pipeline, which varies from 3 billions to 8 billions. The Y-axis is the percentage of machines reduced: $(1 - Cost\_B/Cost\_A)$. $Cost\_A$ is the online cost in terms of the number of machines. $Cost\_B$ includes both online cost and the extra offline cost for the added duplicate analysis service. There are four curves displayed with 4 different configurations and we explain them in details as follows.

In this evaluation, the replication degree of the online system is chosen as 2. The database for the online system is managed in two tiers. The first online tier contains 1.4 billions (marked as T1.4) in one configuration and 2.5 billion (marked as T2.5) in another configuration. The URLs in the first online tier are searched for each query and the second tier is searched only if the results matched from the first online tier are not satisfactory in reaching an internal relevancy score. The selection of online tier 1 data is based on an offline assessment of URL quality such as web link popularity (if there are other web pages pointing such pages) and click popularity (if users have visited such a page). The chance that a duplicate appears in the first online tier is smaller than the second tier, however, there is still a significant percentage of duplicates in the first tier.

Figure 9 indicates that the overall cost saving varies from 8.2% to 25.6%. The total saving increases as the replication of online machines increases to handle more traffic. There are factors contributed this saving using scheme B. 1) Save machines needed for hosting duplicates. Even the second tier receives about 1/3 to 1/4 of total traffic, the total number of machines to host near duplicates in Scheme A is still significantly more than B. 2) Answer more traffic using Tier

1 machines. Because online tier 1 has less near duplicates, tier 1 machines in Scheme B can host more unique content and can answer 5% to 10% more traffic, which in turns saves machine budget in Tier 1. 3) Reduce communication overhead for result collection and improve query handling throughput of the online engine. Since matched candidates in scheme B in answering a query contain more unique results, scheme B collects less matched candidates for ranking from each individual machine compared to scheme A. That significantly reduces communication usage and increases the online engine throughput.

It should be noted that one may consider the reduction of words or position information indexed to save cost. We do not choose this option because of relevancy concerns.

7. CONCLUDING REMARKS

The contribution of this paper is to present our experience in developing a large-scale offline duplicate removal system for processing billions of pages updated continuously. Our parallel clustering scheme is supported by a two-tier architecture for detecting and managing near duplicate groups with incremental computing and multidimensional data mapping. Transitive approximation simplifies duplicate information management, reduces storage space, and avoids certain comparisons.

Our evaluation shows that removing redundant content in an offline system significantly reduces the overall engine cost while sustaining relevancy quality. Incremental update of duplicate groups in this offline two-tier architecture with several approximation techniques greatly speeds up computing time and processing throughput. Multidimensional mapping offers a flexibility in improving load balancing and processing throughput efficiency.

Acknowledgments

We thank Apostolos Gerasoulis, Hong Tang, Rahul Lahiri, Vivek Pathak, Yufan Hu, Honghui Lu, and other Ask.com engineers for their kind support in the earlier studies, and the anonymous referees for their helpful comments. This work was supported in part by NSF IIS-1118106. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Eugene Agichtein, Eric Brill, and Susan Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of ACM SIGIR'06*, pages 19–26.

[2] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 918–929, 2006.

[3] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 131–140, 2007.

[4] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Proc. of Combinatorial Pattern Matching, 11th Annual Symposium*, pages 1–10, 2000.

[5] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Proc. of WWW 1997*, pages 1157–1166.

[6] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002.

[7] Junghoo Cho, Sourashis Roy, and Robert E. Adams. Page quality: in search of an unbiased web ranking. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 551–562, 2005.

[8] Abdur Chowdhury, Ophir Frieder, David A. Grossman, and M. Catherine McCabe. Collection statistics for fast duplicate document detection. *ACM Trans. Inf. Syst.*, 20(2):171–191, 2002.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proc. of OSDI 2004*.

[10] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. A large-scale study of the evolution of web pages. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 669–678, 2003.

[11] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[12] Hannaneh Hajishirzi, Wen tau Yih, and Aleksander Kolcz. Adaptive near-duplicate detection via similarity learning. In *Proc. of ACM SIGIR*, pages 419–426, 2010.

[13] Monika Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proc. of SIGIR '06*, pages 284–291, 2006.

[14] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, 2002.

[15] Aleksander Kolcz, Abdur Chowdhury, and Joshua Alspector. Improved robustness of signature-based near-replica detection via lexicon randomization. In *Proceedings of KDD*, pages 605–610, 2004.

[16] Jimmy Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *SIGIR*, pages 155–162, 2009.

[17] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proc. of WWW '07*, pages 141–150, 2007.

[18] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. *Technical report, Standford Digital Library project*, 1998.

[19] Daniel Peng and Frank Dabek. Large-scale incremental processing using distributed transactions and notifications. In *Proceedings of OSDI'10*, pages 1–15.

[20] Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 743–754, 2004.

[21] Martin Theobald, Jonathan Siddharth, and Andreas Paepcke. Spotsigs: robust and efficient near duplicate detection in large web collections. In *SIGIR '08*, pages 563–570.

[22] Chaokun Wang, Jianmin Wang, Xuemin Lin, Wei Wang, Haixun Wang, Hongsong Li, Wanpeng Tian, Jun Xu, and Rui Li. Mapdupreducer: detecting near duplicates over massive datasets. In *SIGMOD '10*, pages 1119–1122.

[23] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, pages 131–140. ACM, 2008.

[24] Qi Zhang, Yue Zhang, Haomin Yu, and Xuanjing Huang. Efficient partial-duplicate detection based on sequence matching. In *Proc. of SIGIR*, pages 675–682, 2010.