Building a Scalable and Accurate Copy Detection Mechanism

Narayanan Shivakumar and Hector Garcia-Molina
Department of Computer Science
Stanford, CA 94305.
{*shiva, hector*}@*cs.stanford.edu*

Abstract

Often, publishers are reluctant to offer valuable digital documents on the Internet for fear that they will be re-transmitted or copied widely. A Copy Detection Mechanism can help identify such copying. For example, publishers may register their documents with a copy detection server, and the server can then automatically check public sources such as UseNet articles and Web sites for potential illegal copies. The server can search for exact copies, and also for cases where significant portions of documents have been copied. In this paper we study, for the first time, the performance of various copy detection mechanisms, including the disk storage requirements, main memory requirements, response times for registration, and response time for querying. We also contrast performance to the accuracy of the mechanisms (how well they detect partial copies). The results are obtained using SCAM, an experimental server we have implemented, and a collection of 50,000 netnews articles.

1 Introduction

Commercial publishers may be reluctant to offer documents in a Digital Library if their documents can be easily re-transmitted onto UseNet newsgroups, or offered on alternate ftp or Web sites for free. Therefore, a critical problem that needs to be addressed before Digital Libraries are widely used is the prevention or detection of illegal copies (full or partial). A related problem arises in information retrieval from multiple sources [16]. Here many retrieved documents may be duplicates or near duplicates. For example, sources may have copies of the same document, or articles may be cross-posted on several newsgroups. Even a single source may have the same document in different formats (HTML, postscript, DVI, Word) or may have different versions of a document. Here again it would be useful to detect copies, and remove them (if the user wishes) before search results are presented.

In both the above problems, users would like to specify required levels of overlap between documents. For instance, a publisher may wish to be notified even if a small section of his book is available illegally at a certain site. In the rest of this paper, when we refer to "copy detection" we include detection of both full and partial copies. Also, from now on we focus on the detection of illegal copies scenario, although the same principles and algorithms are applicable to the detection of copies in search results.

One way to detect copies is through a *Registration Server* [3, 14, 15]. In this scheme, users (such as authors and publishers) register their valuable digital documents at the server. These documents are "chunked" (broken) into primitive units such as sentences or paragraphs, and these chunks are stored in the repository. Query documents such as netnews articles, and documents available at ftp sites and Web pages are chunked into the same primitive units as the registered documents. These chunks are compared with the set of registered chunks for overlap, and in case of "sufficient" overlap, the owner of the registered copy is notified of the location of possible illegal copies. The registration server can also be used in "reverse:" such a system can register publicly available documents such as netnews articles and Web pages on a daily basis. (The registered documents can be purged from the repository after a few days or weeks.) Authors can then check their documents against this repository for overlap.

Registration servers can be implemented in a variety of ways. The major choices are: (1) the chunking strategy (e.g., small vs. large chunks, overlapping vs. non-overlapping chunks); (2) the data structures used to store chunks and to find matches between registered and queried chunks; and (3) the decision function used to determine when a queried document has "substantial" overlap with a registered one. Our goal in this paper is to evaluate and compare some of these key implementation choices.

One important way to evaluate the choices is to study the *accuracy* of each mechanism, for example, how often does the mechanism report pairs of documents that the user does *not* consider substantial copies, or how often does it miss cases that the user would consider copies? Another very important evaluation criterion is performance: for instance, how long does it take to register documents, what storage and computational resources does the server consume, and how long does it take to check documents against the registered ones? We believe it is very important to study both accuracy *and* performance, since for example, a very efficient mechanism might be preferable to an inefficient one that provides slightly more accuracy.

As far as we know, this paper is the first to study both accuracy and performance. In particular, we address important questions such as: (1) how do chunking strategies impact the registration database size, query response times, and registration times; (2) how does accuracy vary with each of the chunking strategies; and (3) is it effective to remove certain "stop" words from the documents before chunking?

For our evaluations we use an experimental registration server called SCAM (Stanford Copy Analysis Mechanism).[1] SCAM is a flexible system that allows experimentation with the implementation choices outlined above. It also provides a WWW interface and a variety of user services that will be described in Section 2.

We start by reviewing related work (Section 1.1). In Section 2 we describe the SCAM architecture, services and data structures, while in Section 3 we present the chunking strategies that will be studied. In Section 4 we present empirical results on the performance and accuracy of the various strategies.

1.1 Related Work

Protecting digital documents from illegal copying has received a lot of attention recently. Some systems favor the copy *prevention* approach, for example, by physically isolating information (e.g., by placing information on standalone CD-ROM systems), by using special-purpose hardware for authorization [11], or by using *active* documents (e.g., documents encapsulated by programs [7]). We believe such prevention schemes are cumbersome, and may make it difficult for honest users to share information. Furthermore, such prevention schemes can be broken by using software emulators [3] and recording documents. Instead of placing restrictions on the distribution of documents, another approach to protecting digital documents (one we subscribe to) is to *detect* illegal copies using registration server mechanisms such as SCAM [14] or COPS [3]. Once we know a document to be an illegal copy, it is sometimes useful to know the originator of the illegal copy. There have been several proposals [2, 1, 2, 4] to add unique "watermarks" to documents (encoded in word spacing or in images) so that one can trace back to the original buyer of that illegal document.

A variety of mechanisms have been suggested for registration servers. In [9], a few words in a document are chosen as *anchors* and checksums of a following window of characters are computed. "Similar" files can then be found by comparing these checksums that are registered into a database. This tool is mainly intended for file management applications, and detection of files that are very similar, but not for detecting small text overlaps. The COPS [3] and SCAM registration servers however were developed to detect even small overlaps in text. COPS was an early experimental prototype built in our research group that primarily used sentences as its chunking primitive. SCAM is built on similar grounds as COPS with a view on scalability and extensibility of chunking strategies. As mentioned in Section 1, papers on the earlier detection mechanisms only study their accuracy; here we also study their performance.

[1] Readers may be interested to know that recently SCAM was successfully used to track real life instances of plagiarism in several conference papers, and journals [5]. See http://www-db.stanford.edu/šhiva/scamInfo.html for details.)

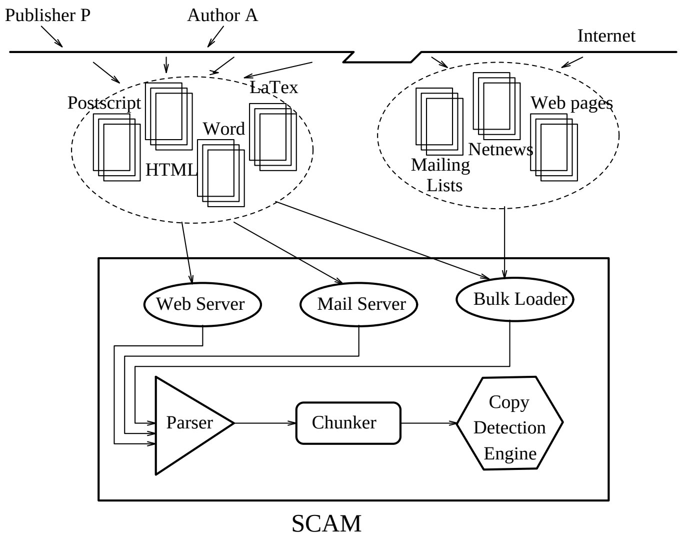

Figure 1: Functionality of SCAM.

All the schemes studied in this paper have at their core a hash table of chunks. For a given chunk, this table reports the registered documents where this chunk appears. There are, however, other ways to compare documents. For example, PAT trees [6] and suffix arrays [8] have been proposed to find maximal common subsequences in documents (with some preprocessing during registration) and are therefore candidates for copy detection. We believe that since these schemes are solving a harder problem (maximal subsequences), they are substantially more expensive than the hashing-based ones. Although we only consider hashing schemes here, we do intend to evaluate other approaches in the future, since they may be useful as "post-filters" to analyze pairs of documents that have been identified by the hash-based schemes.

2 Architecture of SCAM

In this section we first outline the functionality of the SCAM registration server from the user's perspective; subsequently we describe its implementation.

Figure 1 shows the overall functionality of SCAM. SCAM bulkloads documents available on public sources (such as Netnews, Web pages and messages on mailing lists) on a daily basis into its *public* repository. Authors and publishers can then interact with SCAM through its Web Server (by filling out forms) or its Mail server (by sending email). Through these front ends, remote users can create personal repositories of registered documents, or compare documents against SCAM's private or public repositories. SCAM accepts documents (for bulkloading, private registration, or comparison) in several formats (currently HTML, plain ASCII, postscript). All documents are parsed into a common representation (currently ASCII text), and are streamed through the chunker (details in the next section) before the chunks are sent to the Copy Detection Engine for comparison or registration. SCAM reports document overlaps in two ways: (1) users are notified (email messages or Web forms) if overlaps are detected between a query document and some document in the target repository; (2) users are notified if some document they had registered in their private repository overlaps with some document that is later registered in the public repository. Once a user is notified, he can then lookup the document in question to check if the overlap is indeed a legal or ethical problem.

Figure 2 illustrates the main data structures used by the Copy Detection Engine. Keep in mind that our goal is to have structures that can scale to large numbers of documents. For the reader's convenience we have numbered the components shown in the figure, and use these numbers as superscripts in the text that follows. SCAM uses a Buffer Manager (BM)[1] that interacts with the UNIX file system[2] through paged files (blocks of data). The BM caches the following persistent data structures (that reside on disk) into the buffer pool[3] (main memory) for fast access:

- **Inverted Index on Chunks:**[4] In this structure (commonly used in IR systems [12, 13]), we maintain postings indicating the (nonzero) occurrence frequency of chunks in documents. For instance, a posting may be of the form *("picture", 11, 5)* to indicate that the chunk "picture" occurs 5 times in a document with a unique identifier 11. The inverted index consists of an *extensible hashing* structure that is indexed on chunks for efficient lookup. Each chunk points to a set of pages that stores its postings. All the postings for a chunk are clustered together (in the same disk pages) for efficient access, with overflow pages added when necessary.
- **Name Mapper:**[5] In this structure, we store the mapping from the unique document id used in postings to the actual file name (stored locally) or URL of the Web page. For instance, an entry may be of the form *(11, "http://www-db/stanford.edu")* indicating that the document with id 11 is a Web page located at the given URL. This structure is used so that the postings in the inverted index can be smaller by storing integer ids rather than a string of bytes. We implement the name mapper as an extensible hashing structure that is indexed on document id.
- **Relevance Array:**[6] Overlaps between a query document and a set of registered documents is usually kept track by incrementing some "score" (discussed in Section 3) whenever a common chunk is found. For instance, say query document Q has 5 chunks in common with R_1 and 7 chunks in common with R_2, where R_1 and R_2 are some registered documents. When a new chunk is found in common with R_2, the score of R_2 is incremented to 8. Typically such structures that require random accesses to individual data items (like R_2) are implemented as arrays in main memory. However since the set of registered documents could be large, it may not be possible to allocate an array at run time. We implement the relevance array as an extensible hashing structure that is indexed on document id to provide fast access to the relevance scores. Since this structure is also implemented as paged files, as the structure grows, portions can be paged to disk.

We now outline how the above structures are used during the registration and the probe phases.

- **Registration:** When a new document is to be registered, an entry is made in the Name Mapper[5] with the file name of the document or the URL (if it is a Web page). A unique integer is generated that maps to the name of the file. The stream of chunks is then inserted into the main memory based inverted index[7] that keeps track of the document id and the frequency count of the chunk in that document. This inverted index is periodically merged with the persistent inverted index[4]. This way I/O's are batched, reducing the number of random I/Os. To save space, we periodically purge older documents from the public repository, since they are not of much interest.
- **Probing**: When a query document is to be checked against a repository, its chunks are stored into the main memory inverted index[7]. For each chunk stored in the inverted index[7], we retrieve the postings in the persistent inverted index[4] to find the set of documents containing that chunk, and incrementally compute the degree of overlap between the query document and the registered set of documents using the Relevance Array[5]. The user is then notified of the documents that overlap with the query document above some user-specified threshold.

3 Possible Chunking Primitives

In this section we examine different syntactic ways of breaking up documents into more primitive units called chunks. We are primarily concerned with chunks since they form the units of registration/probing and thereby impact decisions concerning document overlap. The chosen units of chunking also influence performance measures significantly. For example, say we compare chunking by words versus chunking by sentences. There are very few words in the English language compared to the number of possible sentences. Hence we expect that, given a certain buffer size, a larger portion of the extensible hash table (of the inverted index) will reside in memory for word chunking. On the other hand, we expect the average number of postings per word to be higher than for sentences, so these longer postings lists will contend for valuable buffer space (in main memory). Depending on the relative impact of the above two effects, one chunking mechanism may dominate the other in terms of disk I/O and thereby throughput and response times. We now explore various chunking alternatives, and in Section 4 we evaluate their performance with a real document set.

No Chunking: A document is a "trivial" chunk of itself. However by not breaking up a document into smaller units it is not possible to detect partial overlaps. This is however very efficient in terms of performance, since to check

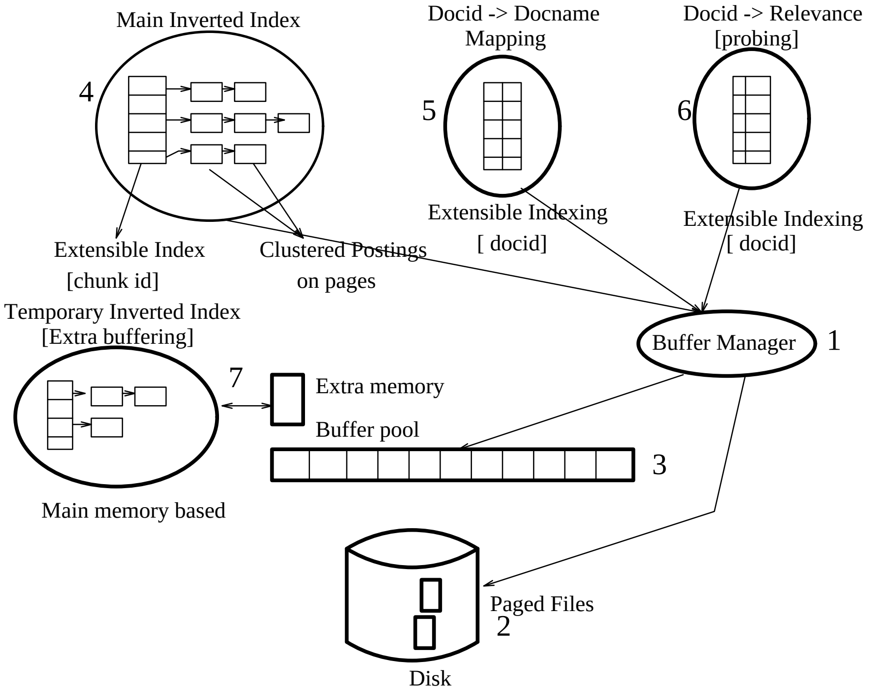

Figure 2: Implementing the Data Structures on the Repository in SCAM.

one document only a single chunk has to be looked up in the inverted index. Also the storage required is low since a hash value of a few bytes (like checksum) suffices to represent each document.

Sentence Chunking: To detect partial overlap between documents, we can reduce the lengths of chunks. If we break up the document into sentences, and store each of the sentences in the repository we will be able to detect document overlaps at the granularity of sentences. A simple metric to quantify the overlap between two documents can be the percentage of shared sentences. This chunking is efficient in disk I/O since only postings of documents that have a common sentence with a query document are retrieved (using our inverted indices). The main disadvantage is that we cannot detect partial sentence overlap using sentences as our chunking primitive. Another important problem is that finding sentence boundaries is a hard problem [3, 10].

Hashed Breakpoint Chunking: To detect overlap at a finer granularity than sentences, [3] has studied various schemes to chunk sequences of words. Brin et al. [3] note that using k consecutive words as a chunking unit leads to a "phasing" problem. For instance, inserting a single word in a document shifts all subsequent chunks by one, making them not match with the original document. Brin et al. also consider using overlapping sequences of k words to avoid this problem but find the storage requirement very high. Brin et al. propose using non-overlapping sequences of words with hashed breakpoints as a compromise that avoids the phasing problem while having low storage costs. This scheme works as follows. Start by hashing the first word in the document. If the hash value modulo k is equal to zero (for some chosen k), the first chunk is merely the first word. If not, consider the second word. If its hash value modulo k is zero, the first two words are considered the chunk. If not, continue to consider the subsequent words until some word has a hash value modulo k equal to zero, and the sequence of words from the previous chunk break until this word will constitute the chunk. A simple metric to determine overlap between two documents will be to compute the number of such shared chunks.

Word Chunking: We may also consider each word to be a chunk. Using standard IR techniques we may then denote two documents to have sufficient overlap if they share a "similar" set of words. However, similar documents do not necessarily have significant textual overlap. To correct for this, the following metric was proposed in [14]; it attempts to identify pairs of documents that are similar because of textual overlap, as opposed to those that are semantically similar. The overlap between documents D_1 and D_2 is defined to be

$$subset(D_1, D_2) = \frac{\sum_{w_i \in c(D_1, D_2)} tf_i(D_1) * tf_i(D_2)}{\sum_{i=1}^{N} \alpha_i^2 * tf_i^2(D_1)} \qquad (1)$$

where w_i is the i^{th} word, $tf_i(D_j)$ is the number of occurrences (term frequency) of word w_i in document D_j. $C(D_1, D_2)$ is the *closeness* set of words which denote the set of words that occur with "similar" frequency in D_1 and D_2. As is done with traditional IR metrics, we can also ignore certain *stop words* in computing the subset metric. In [14] we empirically showed that this measure is in general better for detecting overlap than the classical Cosine measure used commonly in IR [12].

Word chunking loses sequencing information between words (unlike in sentences or sequences of words), but surprisingly it has good accuracy in practice (Section 4). Also since there is a fairly small set of English words (as opposed to sentences), there is more locality in terms of I/Os and hence word chunking has good performance.

4 Experiments

For our experiments we used 50,000 netnews articles from September 29th 1995 as our primary document set. We chose netnews articles since these types of articles "stress-test" copy detection mechanisms for several reasons. Netnews articles are relatively small, so substantial overlap between articles may be a few sentences, making it harder to detect. Also, the articles are informal, so when people copy text into an article, they may copy incomplete sentences. Furthermore, many articles have lengthy "signatures" with character pictures and odd text which often confuses the chunker. We expect that with longer, more conventional documents, copy detection will be easier. Indeed, some preliminary tests we have conducted with conference papers and journal papers confirm this. We do not report these results due to space limitations.

For our netnews document set, in this section we present some empirical performance and accuracy results. We start by defining how accuracy was measured.

4.1 Computing Accuracy

The ultimate goal of a copy detection mechanism is to identify pairs of documents that the human end user would consider to have "substantial" overlap. Thus, for our experiments we can start with a set of documents that are known to have "substantial" overlap, and then see if the computer can correctly identify these. Since the term "substantial" actually depends on the interests or goals of the human, we actually start with three sets of documents.

To obtain these three sets, we proceeded as follows. We examined about 1,000 documents from our registration set in great detail and chose 50 articles that had many partial and complete duplicates. The articles (about 220) that overlapped with the 50 articles were placed into the following sets according to the "degree" of overlap, as judged by the authors:

1. **Exact Copies:** These were articles that were identical due to crosspostings to one of the 50 articles.

2. **High Overlap Replies:** These were articles that responded to the one of the original 50 articles by including most of the text of the original article.

3. **Some Overlap Replies:** This is similar to the previous category, except that only small portions of the original article has been included.

We stress that this classification was very subjective, but was our best attempt to identify documents that most humans would agree were closely related.

4.2 Approximate Computer Tests

The objective of a copy detection scheme is then to identify the articles that were selected in the manual classification. In particular, we studied the following chunking schemes: (1) "Word + 0" chunking where we used word chunking after filtering away words in the WAIS stoplist (391 words), (2) "Word + 100" chunking where we used word chunking after filtering away words in the WAIS stoplist and the 100 most frequent words in the registration document set, (3) "Word + 300" chunking where we used word chunking after filtering away words in the WAIS stoplist and the 300 most frequent words in the registration document set, (4) Sentence chunking, (5) "Modulo 5" which was chunking the document with hashed breakpoints with a modulo 5 hash function ($k = 5$), (6) "Modulo 10," hashed breakpoints and $k = 10$.

Each of these strategies requires a *threshold* to determine when there is substantial overlap. This threshold depends on what type of overlap is targeted. For example, to identify exact copies with sentence chunking, we can look for pairs of documents with 100% identical sentences. If we are looking for documents in our first two test sets (exact and high overlap), we can look for documents that share say 60% of more of the sentences. In [14] we experimented with different thresholds for the various overlap goals, and empirically determined values that worked well in practice. Here we use those thresholds, which are as follows: we set the boundaries between Exact, Lots and Some to be 100%, 66% and 33% for the word chunking schemes, and 100%, 50% and 5% for the Sentence and Modulo schemes.

4.3 Empirical Results for Registration

We had two phases in our experiments: (1) registration and (2) probing. We now outline the registration phase, and will consider the probing phase in the next subsection. We registered the 50,000 netnews articles into SCAM for each of the different chunking primitives. We report on the disk storage required by each of the different chunking mechanisms broken down by the size of the extensible index, the actual data size for postings (size of disk blocks with postings for chunks) and the total disk space (including the Name Mapper array) in Figure 3. We also performed the registrations while varying the buffer pool controlled by our Buffer Manager from 8 MB through 40 MB to observe the influence of buffer size on performance. We report the number of buffer misses (corresponds to number of disk I/Os) and the corresponding bulk load times in Figure 4.

We now consider each of the different performance measures with some of our observations on the results. We first consider the disk storage requirements of our registration server. The original set of 50,000 documents in their complete form consumed about 120 MB of disk space. In Figure 3 we see how much space the different chunking schemes consumed. The two main aspects that affect the required disk space for the different chunking primitives are (1) the number of different chunks, and (2) the number of postings per chunk. As discussed earlier, there are very few words compared to the other forms of chunking, and hence word chunking schemes have very small index sizes. However since

Chunking	Index Size (MB)	Data Size (MB)	Total Size (MB)
Word	4.056	59.528	72
Word (+100)	4.056	51.752	66.5
Word (+300)	4.028	45.028	59
5-Modulo	17.064	53.088	79
10-Modulo	8.048	20.264	37
Sentence	16.200	40.376	64

Figure 3: Disk Storage (in MBs) Required for Diffferent Chunkings.

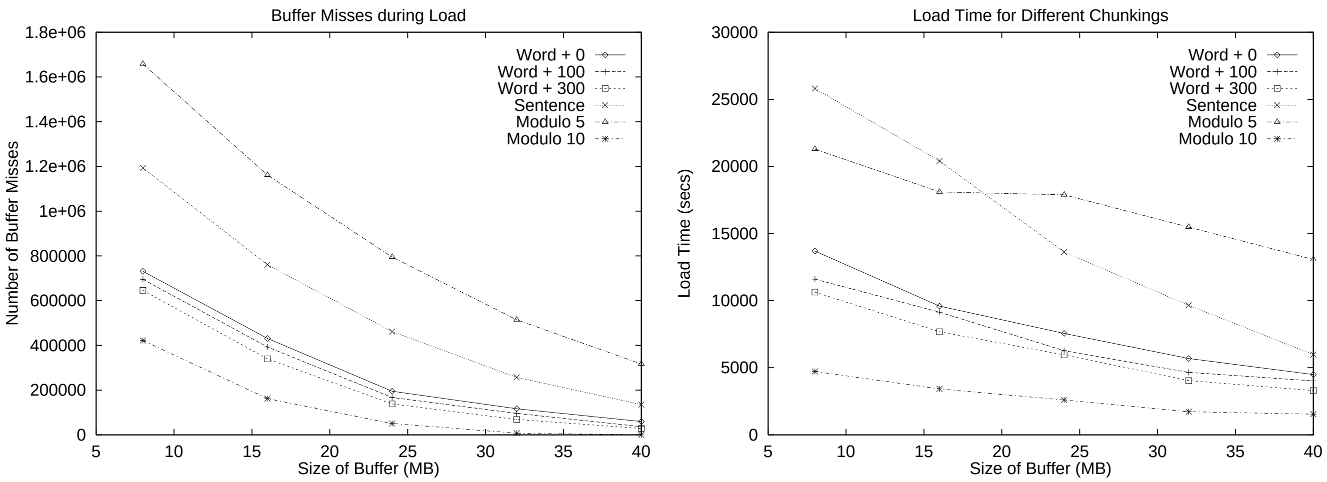

Figure 4: Buffer Misses and Load Time for Different Chunkings.

some popular words occur in several documents, the average number of postings per chunk is high and thereby the data size for postings is higher for word chunkings. Also the data size required for word chunking reduces significantly when we increase the number of stop words since the stop words have many postings. The k-Modulo schemes have an expected chunk length of k [3]. Therefore the 10-Modulo chunking scheme has a very low number of chunks. Since the average number of postings per chunk is also very low (as in sentences), 10-Modulo has the minimal storage requirement. Overall, we see that our repository for the different chunking schemes are about a half to a quarter of the actual data size. Since our data structures scale linearly with the number of documents and since we also store the actual articles, we can store (say) a window of 5-6 days worth of netnews articles in a 1 GB disk before older articles are purged.

We now consider the performance in bulk loading for the different chunking primitives in Figure 4. Clearly the two measures in bulk loading are very related since the number of buffer misses (disk I/O) impact the run time. The two main aspects that impact buffer misses are (1) locality in chunk referencing, and (2) the total disk space consumed by the chunking scheme. The first aspect refers to the number of buffer misses avoided due to the required buffer page already residing in the buffer pool. The second aspect is also critical since the smaller the disk space consumed by a chunking scheme, the higher the percentage of the blocks residing in the buffer pool independent of the locality in chunk access. We see that the word chunking schemes dominate the sentence and the 5-Modulo chunking schemes since they have more locality in chunk access than do the latter even though they consume approximately the same amount of disk space. Also the performance of the word chunking schemes improves with more stop words due to the reduction in disk space. 10-Modulo performs the best despite absence of chunk locality since its low storage requirement helps retain most of the required disk blocks in main memory. Our initial experiments (graphs not shown here) show that load times vary linearly with the number of documents to be registered, confirming that our data structures and loading algorithms scale well.

4.4 Empirical Results for Probing

We now outline the document probing phase in our experiments. We set our buffer pool to be 32 MB and reregistered the 50,000 documents into our registration repository for the different chunking schemes. We probed the sample query set of 50 articles against the registered articles without flushing the buffer pool between probes. For each of the 50 articles, SCAM created three sets S_E, S_H, S_S containing the documents that SCAM denoted to have Exact, High and Some overlap with the article. These sets were compared against H_E, H_H and H_S containing the documents the human denoted to have Exact, High and Some overlap with the same article. Based on this comparison (details below), we report accuracy of overlap detection for each of the different chunkings in Figure 5. We also report on the corresponding response times in Figure 6. We will now explain each of the two figures and how we generated them in great detail.

Chunking	Overlap	Alpha Error	Beta Error
	Exact	0	0
Word + 0	Lots	9.26%	6.66 * 10^{-3}%
	Some	13.88%	0.13%
	Exact	0	0
Word + 100	Lots	6.97%	7.03 * 10^{-3}%
	Some	9.59%	0.09%
	Exact	0	0
Word + 300	Lots	6.89%	2.67*10^{-4}%
	Some	10.12%	7.33*10^{-4}%
	Exact	0	0
5-Modulo	Lots	6.38%	6.79 *10^{-3}%
	Some	11.76%	0.28%
	Exact	0	1.33*10^{-4}%
10-Modulo	Lots	13.04%	2.67*10^{-4}%
	Some	23.88%	6.00*10^{-4}%
	Exact	0	0
Sentence	Lots	12.69%	0
	Some	23.45%	0

Figure 5: Accuracy of Different Chunkings.

We characterize the accuracy of the chunking schemes with alpha and beta errors computed by comparing S_E S_H, and S_S with H_E, H_H and H_S. Alpha errors are false negative errors denoting the documents that SCAM "missed" by either not reporting them as overlaps, or by placing them into a lower category (for instance, placing some document in H_E into S_H rather than S_E). They are computed as a percentage of missed documents to the number of overlapping documents expected by the human. For instance, if a document has 10 overlapping documents in H_E and SCAM reports only 8 of those overlapping documents in S_E, the alpha error is $2/10 = 20\%$. Beta errors are false positive errors for a given article denoting the documents in some S_X for that article, but not in H_X for the same article, where $X = E, H, S$. These errors are computed as a percentage of documents that are false alerts to the total number of registered documents. For instance, with 50,000 registered documents and 50 false alerts, the beta error is $50/50{,}000 = 0.1\%$. We report the averages of these two errors for the 50 query documents in Figure 5 after comparing the SCAM's document overlap clasification against the human classification of documents.

We now consider in detail the accuracy in each of the different chunking schemes in Figure 5. The main factor that impacts accuracy is the average length of chunk. As the length of the chunk increases, it is harder to detect partial overlap since overlapping sequences in two documents may start anywhere within the chunk. Hence longer chunks tend to increase the alpha errors. On the other hand, as the number of chunks increases (due to smaller chunk lengths) the beta error increases due to loss in chunk sequence information. For instance, two documents that share the words "digital" and "library" may really not have any overlap. One way to reduce the beta error, is to increase the threshold for

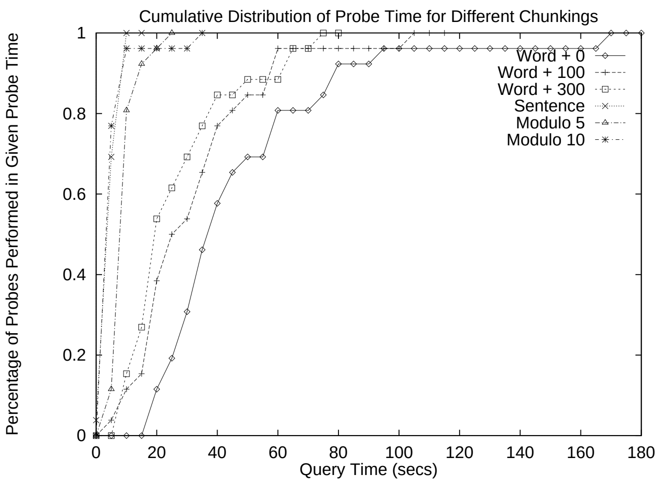

Figure 6: Cumulative Distribution on Probe Time for Documents.

the overlap classifications (S_E, S_H, S_S). But increasing the threshold for overlaps, will increase alpha errors [3]. For instance, if two documents share the phrase "ACM Conference on Digital Libraries" out of 100 words, word chunking may consider 5/100 too small an overlap value (below the chosen threshold) and may miss these two documents with actual overlap. In Figure 5, we see 10-Modulo chunking and sentence chunking as prime examples of "long" chunk lengths which miss a lot of overlapping documents while 5-Modulo chunking performs better. We also see the various word chunkings having high alpha errors due to the choice of thresholds. Another interesting point is that accuracy appears to increase when we increase the number of stop words in word chunking. This is because the number of effective chunks is reduced when we throw away more words, and the less common words cease to be dominated by the more common words in the document overlap computation. Of course we do not expect this trend to continue as more stop words are added since ad absurdum all words will be thrown away and no overlaps can be detected. We intend performing more experiments with stop words in the future.

In Figure 6 we report the cumulative distribution on the probe response times for the 50 documents for each of the different chunking primitives. For example, we see about 35 probes (70%) of the 50 probes take less than a minute to report overlapping documents in the 50,000 document repository for "Word+0" chunking. The main aspects that influence the probe times (a function of buffer misses) are (1) percentage of data residing in buffer pool (function of disk space required for chunking scheme), (2) locality in chunk access, (3) average number of chunks in a document, and (4) average number of postings per chunk. The first factor is important (as in bulk load misses) since there are less likely to be buffer misses if a higher percentage of data is located in main memory. The second factor is important since it leads to a higher chance that a required disk block will be in main memory due to some previous access. The third factor is important since it defines the number of actual buffer accesses that will be made. The fourth factor defines how many possible disk blocks may have to be retrieved when the postings of a chunk needs to be retrieved. The Modulo-5 and the sentence chunking schemes have very low probe times due to factors (3) and (4) dominating (1) and (2). Word chunkings perform relatively poorly since (3) and (4) work against them to a large extent. We see that the word chunking schemes tend to have better performance as the number of stop words increases since they lessen the effects of (3) and (4), and increase the effects of (1) and (2) at the same time. Modulo-10 has factors (1), (3) and (4) working in its favor, and thereby has the lowest probe times.

In summary, from Figures 3, 4, 5 6 we see that there is no single chunking scheme that dominates in all measures of accuracy and performance. Depending on the intended functionality of the copy detection mechanism one chunking primitive may be chosen over another based on our reported results. For instance, 10-Modulo will perform well if the copy detection mechanism needs to detect only Exact copies (so will No Chunking presented earlier). However, we believe that "Word+300" and "5-Modulo" are the more promising chunking primitives. They have the best accuracies among the chunking primitives we presented. Word +300 has the advantage in terms of lower disk requirements and lower bulk load times, while 5-Modulo has the advantage in terms of its low probe times (thereby user response time).

5 Conclusion

We consider how to build an accurate and efficient Copy Detection Mechanism for detecting illegal copies of documents in Digital Libraries. We present the registration repository and the primary persistent data structures that we implemented in SCAM, our experimental prototype. These structures are useful for efficiently finding overlap detection between documents, while retaining the ability to scale to several hundreds of thousands of documents. We outline various chunking primitives and consider their impact on critical performance measures and accuracy. We present empirical results on each of the different chunking primitives, and outline the relative benefits of certain chunking primitives over the others in terms of performance, and accuracy. We show that performance and accuracy vary widely for different chunking mechanisms, making it important to evaluate and understand various chunking options. It is also encouraging that we can indeed build copy detection mechanisms that can scale for large registration repositories. In the future, we intend increasing our registration repository to confirm true scalability. We also intend experimenting more with stop words, the different hashed breakpoint schemes, and schemes based on PAT trees.

References

[1] J. Brassil, S. Low, N. Maxemchuk, and L.O'Gorman. Document marking and identification using both line and word shifting. Technical report, AT&T Bell Labratories, 1994. May be obtained from ftp://ftp.research.att.com/dist/brassil/docmark2.ps.

[2] J. Brassil, S. Low, N. Maxemchuk, and L.O'Gorman. Electronic marking and identification techniques to discourage document copying. Technical report, AT&T Bell Labratories, 1994.

[3] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In *Proceedings of the ACM SIGMOD Annual Conference*, San Francisco, CA, May 1995.

[4] A. Choudhury, N. Maxemchuk, S. Paul, and H. Schulzrinne. Copyright protection for electronic publishing over computer networks. Technical report, AT&T Bell Labratories, 1994. Submitted to IEEE Network Magazine June 1994.

[5] P.J. Denning. Editorial: Plagiarism in the web. *Communications of the ACM*, 38(12), December 1995.

[6] G.H. Gonnet, R.A. Baeza-Yates, and T. Snider. New indices for text: PAT trees and PAT arrays. In *Information Retrieval: Data Structures and Algorithms*, Englewood Cliffs, New Jersey, 1992. Prentice Hall.

[7] G. N. Griswold. A method for protecting copyright on networks. In *Joint Harvard MIT Workshop on Technology Strategies for Protecting Intellectual Property in the Networked Multimedia Environment*, April 1993.

[8] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of 1st ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, California, 1990.

[9] U. Manber and S. Wu. Glimpse: A tool to search through entire file systems. In *Proceedings of the winter USENIX Conference*, January 1994.

[10] D.D. Palmer and M.A. Hearst. Adaptive sentence boundary disambiguation. In *Proceedings of ANLP*, Stuttgart, Germany, October 1994.

[11] G.J. Popek and C.S. Kline. Encryption and secure computer networks. *ACM Computing Surveys*, 11(4):331–356, December 1979.

[12] G. Salton. Term-weighting approaches in automatic text retrieval. *Information processing & management.*, 24(5):513, 1988.

[13] G. Salton. The state of retrieval system evaluation. *Information processing & management.*, 28(4):441, 1992.

[14] N. Shivakumar and H. Garcia-Molina. SCAM: A copy detection mechanism for digital documents. In *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, Austin, Texas, June 1995.

[15] D. Wheeler. Computer networks are said to offer new opportunities for plagiarists. *The Chronicle of Higher Education*, pages 17, 19, June 1993.

[16] T. Yan and H. Garcia-Molina. Duplicate detection in information dissemination. In *Proceedings of Very Large Databases (VLDB'95) Conference*, Zurich, Switzerland, September 1995.