

Parallel and Distributed Document Overlap Detection on the Web

Krisztián Monostori, Arkady Zaslavsky, Heinz Schmidt

School of Computer Science and Software Engineering
Monash University, Melbourne, Australia
{krisztian.monostori, arkady.zaslavsky, heinz.schmidt}
@infotech.monash.edu.au

Abstract. Proliferation of digital libraries plus availability of electronic documents from the Internet have created new challenges for computer science researchers and professionals. Documents are easily copied and redistributed or used to create plagiarised assignments and conference papers. This paper presents a new, two-stage approach for identifying overlapping documents. The first stage is identifying a set of candidate documents that are compared in the second stage using a matching-engine. The algorithm of the matching-engine is based on suffix trees and it modifies the known matching statistics algorithm. Parallel and distributed approaches are discussed at both stages and performance results are presented.

Keywords: copy-detection, string matching, job-distribution, cluster

1. Introduction

Digital libraries and semi-structured text collections provide vast amounts of digitised information on-line. Preventing these documents from unauthorised copying and redistribution is a hard and challenging task, which often results in avoiding putting valuable documents on-line [7]. Copy-prevention mechanisms include distributing information on a separate disk, using special hardware or active documents [8]. One of the most current areas of copy-detection applications is detecting plagiarism. With the enormous growth of the information available on the Internet users have a handy tool for writing or compiling documents. With the numerous search engines users can easily find relevant articles and papers for their research. These documents are available in electronic form, which makes plagiarism feasible by cut-and-paste or drag-and-drop operations while tools to detect plagiarism are almost non-existent.

There are few systems built for plagiarism detection, namely, CHECK [18], Plagiarism.org [17], SCAM [9], Glatt [10], and the “shingling” approach of [3]. They are very similar in their approaches except for Glatt, which does not use computational power but rather assumes that everyone has a different style of writing and users more easily remember their own words rather than plagiarised sentences. The drawback of Glatt’s approach is that users must be involved in the checking process, which makes this approach very cumbersome.

Approaches using computational power include Plagiarism.org, SCAM, and the “shingling” approach. These approaches are common in the way that they are building a special purpose index on documents that reside either on the Internet or in a local repository. There are a few problems when we apply these indexing technics on large data sets. The first issue is finding an appropriate chunking granularity. It is impractical to index all possible chunks of a document. Suffix tree, which is used in this paper, is an index structure that holds all possible suffixes, therefore all possible chunks, of a document. However, suffix trees are too space-consuming to be used as an index [16], though they can be applied when comparing local documents.

Garcia-Molina et al. [9] study document overlap with different chunk sizes. Word chunking is obviously too fine granularity while sentence chunking is not obvious because sentence boundaries are not always easy to detect. Considering a number of words as a chunking primitive is easy to cheat because adding just one-word shifts chunk boundaries. Garcia-Molina et al. [9] propose a so-called hashed breakpoint chunking, which somehow overcomes this problem. The size of the index in the SCAM system is quite large: 30-60% of the size of the original documents depending on the chunking primitive. The “shingling” approach [3] selects chunks to index based on Rabin’s fingerprints. Each ten-word chunk is considered but fingerprinting keeps only every 25th chunk. Using Rabin’s fingerprinting method ensures that selection is random. The storage requirement of this method is more effective: for a 150 GByte data set it takes up only 10GB. The problem with this approach is the elimination of most of the possible chunks but larger chunks still have more chance to be caught because they participate in many chunks.

Because of the extensive space usage of indexes we focus on distributing the index and workload among nodes on the Internet and on a local cluster. Local clusters can be easily built of commodity workstations at low cost. Global clusters are also available by utilising the services provided by Globus [6] and Nimrod/G [1].

In the following sections we discuss our two-stage approach: during the first stage candidate documents are identified using an index and in the second stage candidate documents are compared using the matching engine[15]. Different parallel and distributed approaches of both stages are presented. In Section 2 we outline the algorithm underpinning the matching engine that identifies identical chunks of documents. Section 3 introduces our prototype system, MatchDetectReveal(MDR). Section 4 discusses how we can use the local cluster to analyse documents in the local repository. Section 5 gives a brief overview of the Globus system. Section 6 shows how we can use idle workstations and dedicated servers on the Internet to analyse Internet documents. In section 7 we summarise the work we have done and discuss future work.

2. Identifying Overlap in Documents

The basic problem we address could be summarised in the following way. If given a suspicious document, we want to identify all chunks that overlap with any document either in the local repository or on the Internet. To achieve our goal we use Chang’s matching statistics algorithm [4]. Comparing document P to T the matching statistic

value of $ms(i)$ for P is the length of the longest chunk in T that matches the chunk in P starting at position i . It is obvious that we can identify the overall overlap percentage of document P if we have the matching statistics of P for each document.

The original algorithm builds a suffix tree for T and compares P to this suffix tree. We can use this algorithm in a reverse fashion by building a suffix tree only for P and compare all candidate documents to the suffix tree of P . It is important that the suffix tree is built only once because of potential time and cost saving. By storing some additional information on nodes we can even identify the starting position of a chunk matching another chunk in the suspicious document.

Suffix trees are very versatile [13] but they are very space-consuming data structures. In [16] we proposed a more space-efficient representation of suffix trees, which inserts only suffixes that start at the beginning of words because we are not interested in chunks starting in the middle of a word. Not only does this save space but also the building and matching statistics algorithms benefit from the space reduction towards performance objectives.

Figure 1 depicts the running time of the matching statistics algorithm on the modified suffix tree. We have a single document of 14K and we compare it to sets of documents with different total size. The figure bears out that the algorithm is linear.

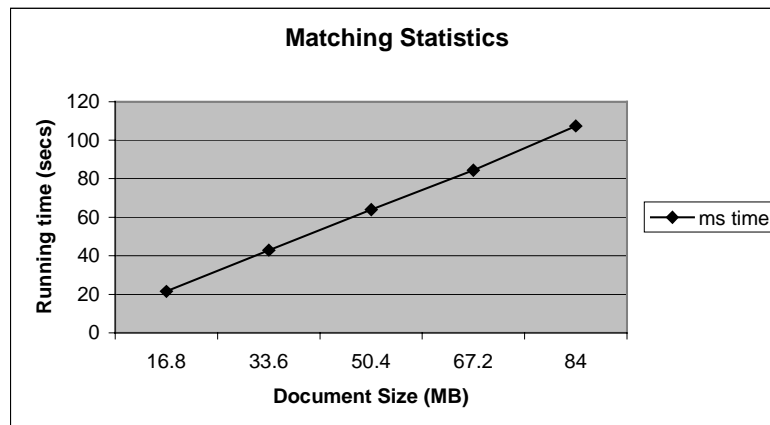


Fig. 1. Running Time of the Matching Statistics Algorithm

3. MatchDetectReveal(MDR) System Architecture

Figure 2 depicts the system architecture of our proposed prototype system MatchDetectReveal [15]. The core of the system is the Matching Engine, which uses the algorithms discussed in the previous section.

Texts must undergo some preprocessing before they can be presented to the Matching Engine. Not only does the Converter component convert documents in different formats (e.g. Adobe Portable Document Format - PDF, PostScript, MS Word etc.) into plain text but it also eliminates multiple whitespaces, converts each

character to lowercase, and shifts some characters, so that the matching algorithm can work more efficiently.

The Matching Engine can only work on a limited number of documents in case of Internet documents because it needs complete documents. Therefore MDR uses a Search Engine, which builds a special-purpose index on the documents from the Internet and only candidate documents identified by the Search Engine are presented to the Matching Engine. Different indexing methods of search engines were discussed in Section 1.

Some additional components provide a user-friendly interface to the system. The Online Submission component enables users to submit documents via a simple Internet browser and results are presented to the user by the Visualiser component.

As depicted in *figure 2* we can use parallel and distributed processing of documents. The following sections discuss how to utilise a local cluster and after a short introduction to the Globus project [6], we propose different extended architectures to utilise idle nodes on a global scale.

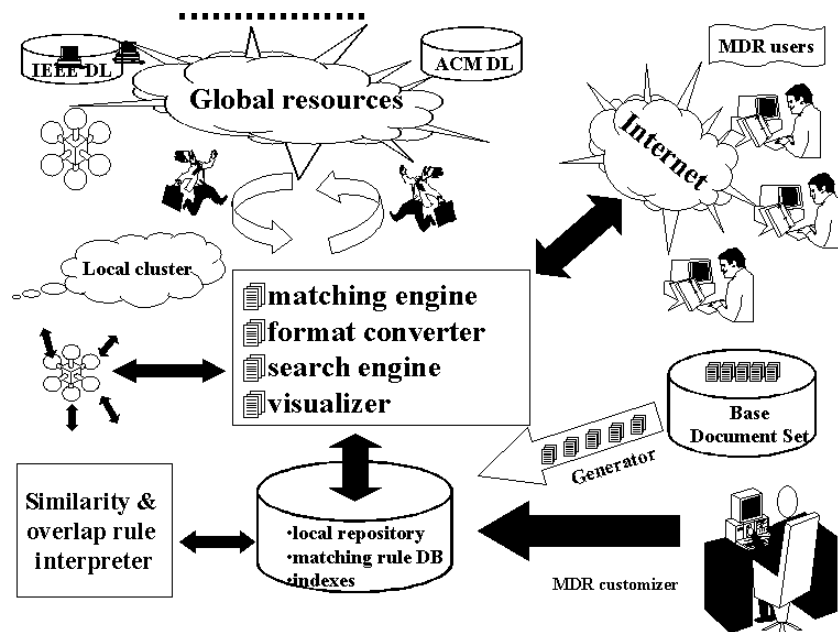


Fig. 2. MatchDetectReveal(MDR) System Architecture

4. Using a Local Cluster to Identify Overlapping Documents

There are several cluster systems built of commodity workstations for different applications including the Berkeley NOW project, the Beowulf project, and Solaris-MC [2]. Most of them provide a Single System Image (SSI), which hides the

complexity of using several nodes. For our experiments we used the Monash Parallel Parametric Modelling Engine. It uses the Clustor tool [5] for transparent distribution and scheduling jobs. There are different ways of utilising a cluster of workstations.

One application is to use it as a final comparison engine in stage two. We identify candidate documents on the Internet, download them, and have different nodes compare them to the suspicious document. When using the Clustor tool we had to pay a time penalty of building a suffix tree for the suspicious document each time a new job is started. Batching candidate documents together, of course, can reduce this time penalty, and Clustor performs well in terms of job scheduling. We only have to provide a plan file, which provides a framework and Clustor does all distribution and scheduling of jobs. If we have one node that downloads documents and distributes them to other nodes using Clustor, the network can easily get congested, which is illustrated in *figure 3*. In this figure we compare the same batch of documents using different number of nodes. The figure shows that using more than 4 or 5 nodes does not add significant speed-up.

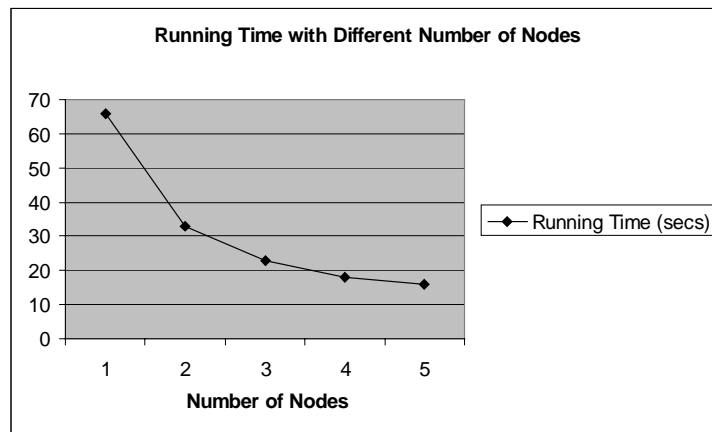


Fig. 3. Running Time Using Different Number of Nodes

If we use a local repository we can have a dedicated server storing the local repository and distribute comparison jobs using Clustor. In this case the same network-congestion problem comes up. If we bypass Clustor and use our own job-scheduling algorithm we can store the repository distributed on nodes. In this case we only have to distribute the suspicious document or its suffix tree and nodes can primarily compare documents on their local disks and when they are finished with local documents they can request documents from other nodes for comparison. No matter which approach we use we have to merge results from different nodes. Fortunately the matching statistics values are easy to merge: it is a simple maximum on each element of the matching statistics array.

5. The Globus Project

In this section we briefly introduce the Globus project [11]. The Globus project aims to provide a general metacomputing infrastructure that can connect heterogeneous environments and act like a virtual supercomputer. Networked virtual supercomputers can both increase utilisation of resources and use unique capabilities of separate resources that otherwise could not be created effectively [6]. Three significant testbeds currently running include National Technology Grid by NPACI and NCSA, NASA Information Power Grid (IPG), and the Globus Ubiquitous Supercomputing Testbed (GUSTO). The Monash Parallel Parametric Modelling Engine is part of the latter one. *Figure 4* shows the integrated grid architecture.

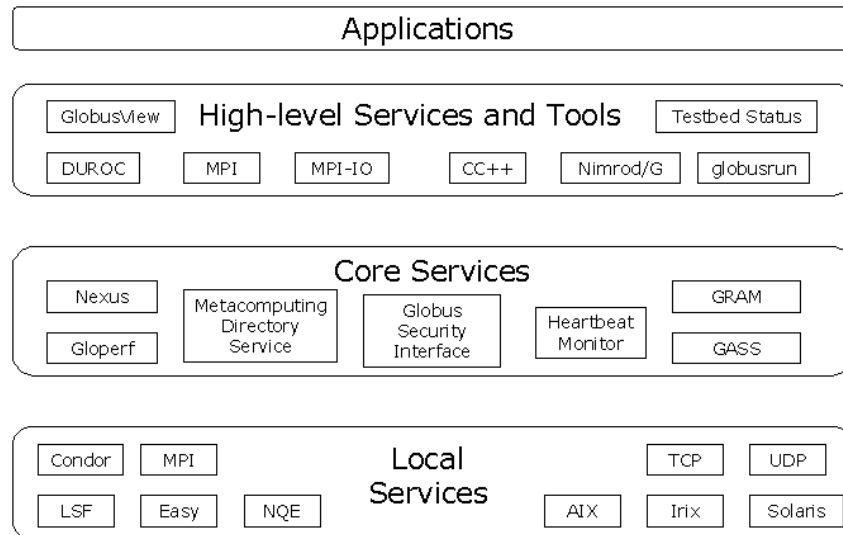


Fig. 4. Globus Architecture

At the lowest level there are local services, i.e. local supercomputers, clusters, or individual computers with their own operating systems, network connections, network libraries etc.

Core Grid services are built on top of local services to provide a uniform interface for higher-level components. Some important core services are detailed in the following subsections.

High-level services use core services to provide an interface for applications, which sit at the highest-level of the architecture. One of these high-level tools is Nimrod/G [1], which is a Cluster-like tool for the global grid rather than the local cluster. Nimrod/G uses a simple cost formula and allocates resources in a way that minimises cost while meeting time constraints. MPICH-G is the Globus version of MPI, which allows users to write applications for the global grid using the standard Message Passing Interface [12].

In the following subsection we briefly describe the three most important core services, i.e. Global Resource Allocation Manager, Grid Information Services

(formerly known as Metacomputing Directory Service), and Grid Security Infrastructure.

5.1 Global Resource Allocation Manager (GRAM)

GRAM is a basic library service that enables remote job submission. There are specific application toolkit commands based on the top of this service. For example the `globus-job-run` command uses an RSL file (Resource Specification Language) to describe which resources to allocate and which jobs must be executed on those resources.

5.2 Grid Information Services (GIS)

The GIS stores information about the state of the grid. Information available about resources includes host name, type of operating system, number of nodes, available memory, etc. The GIS is capable of publishing information via LDAP (Lightweight Directory Access Protocol). It functions both as “white pages” and “yellow pages” directory. The current status of resources are regularly updated, so adaptive applications might for example use a different communication method or even choose another resource if some parameters change during execution.

5.3 Grid Security Infrastructure (GSI)

The GSI provides generic security services for Grid-enabled applications. Each site may apply different security mechanisms, which would make it difficult for a Grid-application to authenticate itself to each resource. Rather than prepare the application for different types of authentication, it can use the API or high-level commands of GSI, and request a Globus certificate. Then it only have to provide this certificate to authenticate itself for a given resource.

6. Identifying Overlap Using Internet Resources

In Section 1 we outlined the storage requirements of an appropriate index for overlap detection. At today’s growth-rate of the Internet it is hard to store and browse such a large index. One option is to partition the index and store these partitions on a local cluster. Hence on comparison each node can analyse documents against their own index partitions and identify their own sets of candidate documents. These sets are then merged and candidate documents are analysed by the matching-engine. The matching process can be either distributed on the local cluster as discussed in the previous section, or we can use idle resources on the Internet to execute the final comparison task using the resource discovery capabilities of Globus.

Using GIS we can assign jobs to resources that are actually in the close proximity of the document to be analysed and downloading time can be reduced in this way.

Not only can our distributed index reside on a local cluster but it can also be distributed on the Internet. We can have servers in different subnets of the Internet, which are responsible for indexing their own subnets and comparison jobs can also be executed on those nodes. Results of these comparisons, that is sets of candidate documents, are collected and the final matching process can be either executed on the same nodes or scheduled to be run on different nodes.

There have been some efforts to use the Grid for data sharing [14], which may also be applied in our system. There could be a testbed set up for copy-detection. Each participating organisation (universities, digital libraries, etc.) would publish resources in this testbed for copy-detection jobs. A copy-detection task could be divided into jobs submitted to separate resources that would compare suspicious documents to their local documents.

7. Conclusion and Future Work

In this paper we discussed different overlap-detection schemes. We found that the storage and time requirement of identifying overlapping documents are large enough to consider parallel and distributed approaches. We introduced a two-stage process. In the first stage candidate documents are identified while in the second stage candidate documents are compared using our matching-engine. We showed how these two stages can be parallelised and distributed both on a local cluster and on the Internet.

Performance results are presented using the local cluster deployed in our school. They show that the Clustor tool loses efficiency when processing data-intensive jobs, so we are implementing our own job-distribution scheme to achieve better performance. We are also currently analysing the efficiency of distributed approaches and implementing a plagiarism-detection prototype named MatchDetectReveal (MDR).

Acknowledgment

Support from Distributed Systems Technology Centre (DSTC Pty Ltd) for this project is thankfully acknowledged.

References

1. Abramson, D., Giddy, J. and Kotler, L. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?. International Parallel and Distributed Processing Symposium (IPDPS), pp 520- 528, Cancun, Mexico, May 2000.
2. Baker M., Buyya R. Cluster Computing at a Glance in Buyya R. High Performance Cluster Computing. (Prentice Hall) pp. 3-47, 1999.
3. Broder A.Z., Glassman S.C., Manasse M.S. Syntactic Clustering of the Web. *Sixth International Web Conference*, Santa Clara, California USA. URL <http://decweb.ethz.ch/WWW6/Technical/Paper205/paper205.html>

4. Chang W.I., Lawler E.L. Sublinear Approximate String Matching and Biological Applications. *Algorithmica* 12. pp. 327-344, 1994.
5. Clustor Manual (1999). URL <http://hathor.cs.monash.edu.au/clustor/>
6. Foster I., Kesselman C. Globus: A Metacomputing Infrastructure Toolkit. *Intl J Supercomputer Applications* 11(2), pp. 115-128, 1997.
7. Garcia-Molina H., Shivakumar N. (1995a). The SCAM Approach To Copy Detection in Digital Libraries. *D-lib Magazine*, November.
8. Garcia-Molina H., Shivakumar N. (1995b). SCAM: A Copy Detection Mechanism for Digital Documents. *Proceedings of 2nd International Conference in Theory and Practice of Digital Libraries (DL'95)*, June 11 - 13, Austin, Texas.
9. Garcia-Molina H., Shivakumar N. (1996a). Building a Scalable and Accurate Copy Detection Mechanism. *Proceedings of 1st ACM International Conference on Digital Libraries (DL'96) March, Bethesda Maryland*.
10. Glatt Plagiarism Screening Program. URL <http://www.plagiarism.com/screen.id.htm>, 1999.
11. Globus Quick Start Guide. URL <http://www.globus.org/toolkit/documentation/QuickStart.pdf>
12. Gropp W., Lusk E., Skjellum A. (1994). Using MPI. Portable Parallel Programming with the Message-Passing Interface. (The MIT Press)
13. Gusfield D. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. (Cambridge University Press), 1997.
14. Kesselman C. Data Grids for Next Generation Problems in Science and Engineering. In this proceedings.
15. Monostori K., Zaslavsky A., Schmidt H. MatchDetectReveal: Finding Overlapping and Similar Digital Documents. *Proceedings of IRMA International Conference, Anchorage, Alaska, 21-24 May, 2000*.
16. Monostori K., Zaslavsky A., Schmidt H. Parallel Overlap and Similarity Detection in Semi-Structured Document Collections. *Proceedings of 6th Annual Australasian Conference on Parallel And Real-Time Systems (PART '99)*, Melbourne, Australia, 1999.
17. Plagiarism.org, the Internet plagiarism detection service for authors & education URL <http://www.plagiarism.org>, 1999.
18. Si A., Leong H.V., Lau R. W. H. CHECK: A Document Plagiarism Detection System. *Proceedings of ACM Symposium for Applied Computing*, pp.70-77, Feb. 1997.