SE4SEE: A Grid-Enabled Search Engine for South-East Europe

B. Barla Cambazoglu, Ata Turk, Evren Karaca,
Cevdet Aykanat, Bora Ucar, and Tayfun Kucukyilmaz
Computer Engineering Department
Bilkent University
Bilkent 06800, Ankara, Turkey
Phone: +90 312 290 1625 Fax: +90 312 266 4047
E-mail: berkant@cs.bilkent.edu.tr

Onur Temizsoylu
TÜBİTAK, ULAKBİM, YÖK Binası B5 Blok
Bilkent 06539, Ankara, Turkey

Abstract - Search Engine for South-East Europe (SE4SEE) is an application project aiming to develop a grid-enabled search engine that specifically targets the countries in the South-East Europe. It is one of the two selected regional applications currently implemented in the SEE-GRID FP6 project. This paper describes the design details of SE4SEE and provides an architectural overview of the application.

I. INTRODUCTION

The ever-growing size of the Web when coupled with the number of users querying the Web pages requires the use of state-of-the-art search engines [1], [2] for accessing valuable information. In the last decade, an explosion was observed in the number of publicly usable search engines. These engines varied in many aspects, including coverage over the pages (e.g., the whole Web, topic-specific, or language-specific), geographical locality, provided user interface (e.g., keyword-based or directory-based), architectural design issues employed (i.e., indexing, compression, filtering), and the way the results are presented to users.

Although many others exist, page coverage and freshness are the most important challenges that a search engine must face. Among these, page coverage refers to the amount of indexed Web pages reachable by user queries, and page freshness is a measure of how up-to-date the indexed content is relative to the original copy in the Web. Higher coverage allows users to access more pages, thus improving recall. Similarly, high freshness may be said to be increasing the relevance of the returned pages, thus improving precision.

Considering the fact that the number of Web pages is in the order of billions and it takes more than a month to fully index the Web, even with the most powerful parallel computing systems, it is seen that both coverage and freshness are outstanding problems before general-purpose search engines. A solution, which recently became popular to these problems is personalized or focused search. In this approach, an on-demand and dynamic search is started on the original copies of pages in the Web. Pages are retrieved in a focused manner by following the hyperlinks within the pages. The relevance of a page to what the user requested is evaluated by some metric. This type of search explicitly solves both coverage and freshness problems. The coverage problem is no longer an issue since the user is free to retrieve as many pages as he requests. Freshness is not an issue since the search is performed over the original copies of pages. The disadvantage of personalized search is that the required computational resource (e.g., CPU, volatile memory, disk storage, network) capacities are quite high.

In this paper, we present the design details of a grid enabled, personalized search engine (SE4SEE), one of the two regional applications [3] in SEE GRID [4], a European Union FP6 project. Since it is a regional application, SE4SEE specifically targets the Web pages located in the countries of South East Europe. The application is composed of two components, a Web crawling component for retrieving pages and a categorization component for evaluating relevance of retrieved pages. This paper provides the design details of these two components and discusses further grid-related issues.

The outline of the paper is as follows. Sections II and III respectively provide a brief overview about Web crawling and text categorization. Then, in Section IV, we make a brief introduction to Grid computing. We present the architectural components of SE4SEE in Section V. Finally, we point at some future work and conclude in Sections VI and VII, respectively.

II. WEB CRAWLING

Web crawling [5], [6], [7], [8] is the process of locating, fetching, and storing a set of pages in the Web. A typical Web crawler, starting from a set of seed pages, locates new page URLs by parsing the downloaded pages and extracting the hyperlinks within. Extracted URLs are stored in a FIFO fetch queue for further retrieval. Crawling continues until the fetch queue gets empty or a satisfactory number of pages is downloaded. Usually, many crawler threads execute concurrently in order to overlap network operations with CPU processing, thus increasing the throughput.

In personalized search, since only a limited subset of the pages is requested, it is not wise to crawl every page. A more clever approach is to analyze page contents and insert into the fetch queue only those pages that are highly relevant to the user request. The basic idea behind this is that if a page is relevant, then its links are also likely to be relevant. This type of a focused crawl has the benefits of reducing the crawl time and increasing the quality of the returned pages in terms of precision.

However, deciding the relevance of a page is a challenging task. In the literature, most works cast this problem as a machine learning problem. In this work, we use text categorization techniques to measure the degree of relevance of pages to user queries. The next section provides some background on text categorization.

III. TEXT CATEGORIZATION

Text categorization [9], [10], [11] is the task of assigning a category to a given text document by analyzing the attributes of the textual material contained in the document. In other words, it can be defined as the process of choosing a type or topic for a piece of text among a predefined set of types or topics. Text categorization, as other text processing problems such as topic identification, text summarization, and text clustering are, is a rather hard problem to be solved. At the extreme case, a mixture of natural language processing and artificial intelligence techniques, which perform semantic and contextual analysis, is necessary for accurate categorization of text documents.

However, in this work, the focus is on statistical techniques [12], which rely solely on syntactical analysis and were abundantly used for categorization in the past. In the literature, different techniques are proposed as solution to the text categorization problem. These solutions are mainly based on application of different machine learning algorithms on text categorization. Applications of k-nearest neighbor [13], [14], naïve Bayesian [15], neural networks [16], and decision trees [17] are among the most popular examples.

The main reasons which prevent further improvement on both efficiency and accuracy of the algorithms mentioned above result from the nature of textual data. As pointed out in many works, the basic reason is the high dimensionality of the attribute space of a document and the high amount of sparsity in documents' attribute spaces. In other words, the number of distinct terms that may occur in a document data set is in the order of ten thousands, but only a small fraction of these terms occur in a single document. In most works, this high-dimensionality problem is attacked and eliminated within a special preprocessing step. This preprocessing step mainly contains either all or some of the following techniques: stopword elimination, stemming, word grouping, and feature selection [18]. In the implementation of the SE4SEE application, we try to incorporate these techniques into the text categorization component as much as possible.

IV. GRID COMPUTING

Grid is a type of parallel and distributed system that enables sharing, selection, and aggregation of geographically distributed autonomous resources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements [19]. According to the functionality they provide, grid systems may be classified as computational, data, or service grids. Computational grids are usually made up of high-performance servers, which together supply a vast computing power to the users. Data grids store large amounts data distributed across multiple computers and aim to let the users from different organizations share and manage the data. Service grids are systems that provide software services that cannot be provided by a single machine.

As we stated in Section I, the computational requirements for personalized Web search are quite overwhelming for a traditional PC. To begin with, a high amount of processing power is needed to parse the crawled page contents, extract the hyperlinks, and categorize the pages. Along with processing power, large amounts of volatile memory is needed to manage the data structures, which continuously and quickly grow during the crawl. If the page download rate is higher than the throughput of the text classifier, secondary storage may also be needed to temporarily store the downloaded pages. The final and probably the most important requirement is on the network bandwidth. The network bandwidth affects the duration of the search session and hence indirectly affects the coverage. All these requirements make personalized Web searching a suitable target for grid computing.

V. SE4SEE ARCHITECTURE

A. Overview

The basic goal of the SE4SEE application is to let the users have an environment in which Web pages can be searched and gathered in an on-demand and focused manner, thus ensuring the freshness of the gathered documents. The underlying structure of the proposed work consists of a crawling and a categorization component. The user queries involve a category and a country name. Upon receiving a query, the crawling component, starting from a predetermined set of seed pages, is made responsible for gathering the pages that match the user's requests from the Web. The downloaded pages are categorized by the text classifier, and the resulting set of pages is returned to the user. The set of categories (e.g., news, festivities, sales, etc.) are predetermined. The application requires language-specific tools for categorization, a set of initial seed pages for crawling each category, and a set of training documents to build a text classification model.

Fig. 1 depicts a sample search scenario over the SE4SEE architecture. In the figure, there are six SE4SEE servers, two located in Bulgaria, three in Turkey, and one in Croatia. The specialized Web portal resides in Ankara, Turkey. Authenticated clients from any of the participating countries use Java enabled Web browsers to connect to the Web portal and submit queries. For example, in the figure a user from Romania wants to find the hotels in Croatia and thus submits this query to the Web portal. The portal then locates an available grid node in Croatia that will act as a SE4SEE server and sends a job description to it. The SE4SEE server initiates a crawling job and collects the URLs of the hotels in Croatia. The collected URLs are then stored in the output sandbox. Upon user's request, the sandbox is retrieved and displayed by the Web portal.

B. Web Portal

In order to effectively provide a Web-based access to the SE4SEE infrastructure and to make the application available to a large number of users, deployment of a Web portal is necessary. This Web interface should conceal the details of the underlying grid infrastructure along with the technologies used. The Web portal should also provide a transparent and user-friendly interface to the non-expert user, as the target audience is not guaranteed to be technology-savvy.

The Web portal is currently being developed using the GridSphere portal framework [20]. The functionalities that will be provided within the portal include authenticating users, accepting user queries from clients, submitting que-

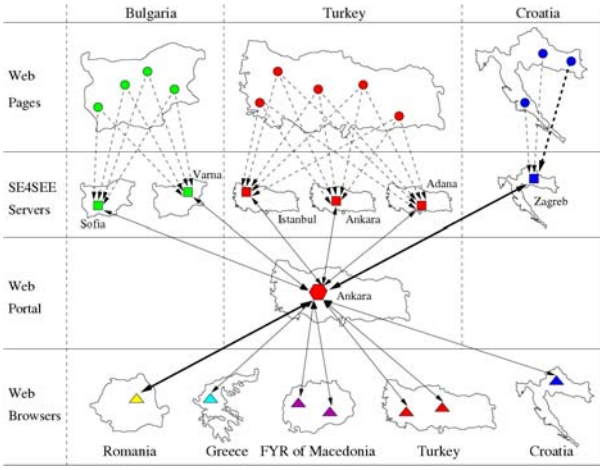

Fig. 1. A search scenario over the SE4SEE architecture.

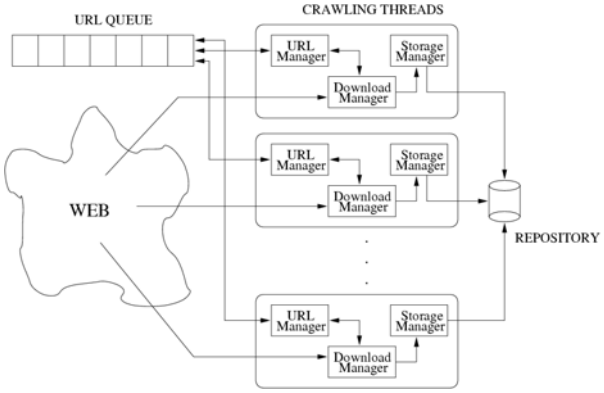

Fig. 2. The Web crawling component.

ries to SE4SEE servers, and collecting the outputs gathered from the servers in order to display the results to clients. The user will be able to select from a number of service-providing sites to submit its search query. The underlying operations performed by the Web portal are to communicate with MyProxy server in order to authenticate users, to generate a job description in JDL (job description language) for user queries, to submit the job descriptions to UI (user interface) nodes of the selected sites, to monitor the submitted jobs, and finally to collect the outputs by using the EDG (European Data Grid) utilities in the LCG 2 infrastructure [21]. Since users may not always harvest the collected outputs immediately, a file proxy server must keep the results for a period.

C. SE4SEE Servers

The servers of the SE4SEE applications run the LCG 2 infrastructure to initiate crawling jobs for the user queries. The crawling jobs require seed pages in order to start crawling. These seed pages need to be country- and category-specific, and hence, they are provided to the SE4SEE servers by the Web portal in the input sandbox. The crawled pages are then categorized. In order to effectively categorize the collected pages, the server requires language-specific categorization tools such as stop-word eliminators and stemming tools. These tools are provided either by the Web portal in the sandbox of the job description or by using GridFTP if their sizes are very large. The URLs that fall into the user-specified categories after the completion of the crawling and categorization jobs will be posted to the Web portal in the output sandbox.

D. Web Crawler Component

This component, developed using the WebSphinx Web crawler library [22], performs the crawling task. The general architecture for the crawler is depicted in Fig. 2. In general, there are several threads running concurrently. Each thread is composed of three main parts: a download manager, a URL manager, and a storage manager. The download manager is responsible for locating the DNS entries for the pages and fetching them from the Web. Each downloaded page is passed to the URL manager. The URL manager parses the pages and extracts the new URLs contained within them. The URLs pointing at pages that are not previously fetched are stored in a common URL queue, accessible by all threads. The URL manager is also responsible for providing new URLs to the download manager for retrieval. The storage manager saves the pages supplied by the download manager on the disk.

The WebSphinx API provides methods to set the crawling parameters such as the seed pages, maximum size of crawled pages, characteristics of the links that can be followed, and the type of the Web graph traversal technique that will be used. For a crawling request, which is in the form of a country and category pair, a crawler is initiated in the queried country that uses the category's country-specific seed pages as its starting point. It is important to assign a query for a user-specified country to a worker node located in the target region. This approach allows geographically close Web pages to be downloaded faster and hence uses the available network bandwidth more effectively and efficiently. Thus, this scheme suits well to grid computing and the selected regional application.

E. Text Categorization Component

Fig. 3 depicts the general picture of the input-output relation among the modules of the classification system that has been designed and implemented. In the figure, an ellipse corresponds to a module or more specifically a piece of code, which can be compiled and executed independently. Solid oblong boxes represent the files stored on the disk. The arrows on the arcs between the files and the modules indicate whether the file is supplied as input to a module or generated by it. Dashed boxes are the inputs passed as parameters to a module during the initial module startup. Bold lines indicate user interaction. The corpus creator and corpus parser modules are the two preprocessing modules used prior to text categorization. They are shortly described below:

Corpus creator: The aim of the corpus creator is to transform a given document collection into a common and standard format. This module performs all text filtering tasks on the given collection, i.e., it eliminates white spaces and removes punctuation from the text. The extracted alphanumeric character groups are converted into upper case and written into a single, formatted corpus file. Since the collection of input documents can be unformatted and vary in size and structure, it may be necessary to modify the I/O routines of this module depending on the

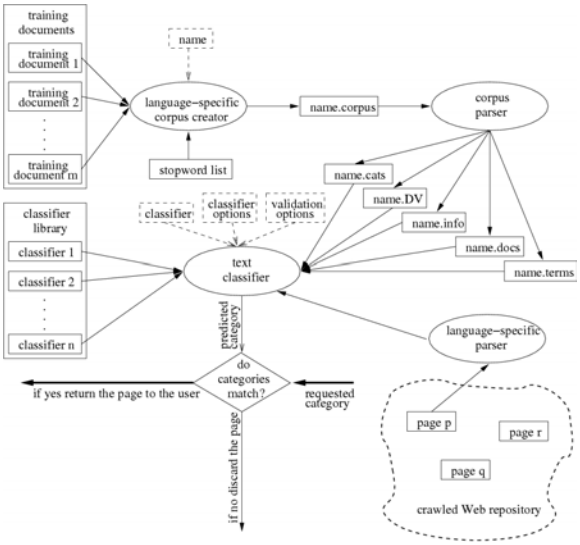

Fig. 3. The text categorization component.

input's properties. Hence, it is preferred to use a separate corpus creator module for each document collection at hand.

Corpus parser: Once the collection is converted into a standard corpus format, the corpus parser module can be used to generate the files that keep detailed information about the corpus. The corpus parser module reads a single corpus file and produces five output files. These newly generated files are all in ASCII format and are used by the text classifier. The corpus parser module is able to apply some cleansing procedures on the input document corpus. If a stop-word file is available in the working directory, the stop-words are eliminated. In this module, it is also possible to remove very short or long terms, discard completely numeric terms, and apply stemming on the terms.

Text classifier: The core of the classification system is the text classifier module. This module is a wrapper, which calls the appropriate machine learning classifiers for the text classification task. It is also possible to run each classifier as a stand-alone application. This module offers several validation techniques and hides the details of partitioning the training document set. The classifier to be used and its options are passed as input to the module. The wrapper first reads the document collection for training purposes and generates a classification model. It then consecutively reads the documents whose categories are to be predicted from the disk and tries to guess a category for each document using the classifier chosen from the classifier library.

Classifier library: As the machine learning library, Harbinger Machine Learning Toolkit [23] is used. The classifiers supported by this toolkit (currently around 10) are collected under four main headings: instance-based classifiers, probabilistic classifiers, symbolic learning classifiers, and neural network classifiers. A more thorough discussion of classifier options, file formats, and several examples can be found in the Harbinger Machine Learning Toolkit manual [23].

F. Client Side

There are no special hardware and software requirements necessary at the client side apart from Java and a Web browser. Executables and required data files are prepared and then sent through the job input sandbox by the Web portal. The user is provided with an interface that allows selection from available grid nodes and specification of the crawling country, crawling category, crawling time, and many additional crawling and categorization parameters. The output is displayed as a list of URLs that fall into the category and crawling specifications given by the user.

G. Further Grid Considerations

On-demand crawling and categorization is a time-consuming task. The produced results may not be available in a short amount of time. Hence, a proxy is necessary on the Web portal. The purpose of this proxy is to mediate between the user and the SE4SEE computing elements. This allows the user to harvest the outcome of the search process, i.e., the gathered set of pages in the future. Another issue is monitoring the status of a submitted search task. It is important to notify the user about the status of such a long-lasting job. We plan to use the EDG features from LCG 2 for monitoring.

VI. FUTURE WORK

In the current implementation, the downloaded pages are temporarily stored for the sole purpose of categorization. A possible extension may be to provide a cache keeping previously obtained sets of URLs in order to provide faster and direct access to highly requested pages. Another extension may be to collect several user queries and submit them as batch jobs. This may allow the sharing of crawled pages among different search tasks. Yet another extension is to provide asynchronous and incremental update to query results. The query results may have a certain life-time (at most certificate life-time). The SE4SEE servers may send newly crawled results to the Web portal, where the previous results are merged.

VII. CONCLUSIONS

The primary goal of the SE4SEE application is to produce a category-based, on-demand search facility that offers access to the set of Web pages located in the South East Europe region in an efficient and user-friendly fashion. Moreover, this application aims to form the necessary infrastructure and lead the way to a general-purpose search engine that considers the needs of the South East European citizens. However, the ultimate goal of the SE4SEE application is to enhance dissemination of information and socio-cultural integration among the countries in the region by providing access to a common knowledge base. Finally, it must be stated that, the categorized data that will be obtained as the outcome of the application is of great importance to many scientific disciplines. It is possible to conduct statistical, sociological, and physiological experiments on the collected Web data.

ACKNOWLEDGEMENTS

This publication is based on the work performed in the framework of the FP6 project SEE-GRID, which is funded by the European Community. The SEE-GRID consortium consists of eleven contractors: ten representatives or incubators of National Grid Initiatives (NGIs) from SE European countries and CERN. The consortium contractors that represent NGIs are: GRNET (Greece), SZTAKI (Hungary), ICI (Romania), CLPP (Bulgaria), TUBITAK (Turkey), ASA (Albania), BIHARNET (Bosnia Herzegovina), UKIM (FYRoM), UOB (Serbia-Montenegro), RBI (Croatia).

REFERENCES

[1] L. Page and S. Brin, "The Anatomy of a Large-Scale Hypertextual Web Search Engine", *Proceedings of the Seventh World-Wide Web Conference*, pp. 107–117, 1998.

[2] Google homepage, http://www.google.com/

[3] SEE-GRID Project Deliverable D3.1: Specifications of Regional Applications

[4] SEE-GRID homepage, http://www.see-grid.org/

[5] B.B. Cambazoglu, A. Turk, and C. Aykanat, "Data-Parallel Web Crawling Models", *Lecture Notes in Computer Science*, vol. 3280, pp. 801–809, 2004.

[6] A. Heydon and M. Najork, "Mercator: A Scalable, Extensible Web Crawler", *World Wide Web*, vol. 2, no. 4, pp. 219–229, 1999.

[7] J. Cho and H. Garcia-Molina, "Parallel Crawlers", *Proceedings of the Seventh World-Wide Web Conference*, pp. 124–135, 2002.

[8] V. Shkapenyuk and T. Suel, "Design and Implementation of a High-Performance Distributed Web Crawler", *International Conference on Data Engineering*, pp. 357–368, 2002.

[9] Y. Yang and X. Liu, "A Re-examination of Text Categorization Methods", *Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 42–49, 1999.

[10] M. Grobelnik and D. Mladenic, "Efficient Text Categorization", *Text Mining Workshop on ECML-98,* 1998.

[11] W. Lam, M. E. Ruiz, and P. Srinivasan, "Automatic Text Categorization and Its Applications to Text Retrieval", *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 6, pp. 865–879, 1999.

[12] Y. Yang, "An Evaluation of Statistical Approaches to Text Categorization", *Journal of Information Retrieval*, vol. 1, no. 1/2, pp. 67–88, 1999.

[13] E. Han, G. Karypis, and V. Kumar, "Text Categorization Using Weight Adjusted k-Nearest Neighbor Classification", *Proceedings of the 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 53–65, 2001.

[14] T. Yavuz and H.A. Guvenir, "Application of k-Nearest Neighbor on Feature Projections Classifier to Text Categorization", *Proceedings of the 13th International Symposium on Computer and Information Sciences*, pp. 135–142, 1998.

[15] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification", *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[16] H.T. Ng, W.B. Goh, and K.L. Low, "Feature Selection, Perceptron Learning, and a Usability Case Study for Text Categorization", *Proceedings of the 20th International Conference on Research and Development in Information Retrieval*, pp. 67–73, 1997.

[17] D.D. Lewis and M. Ringuette, "A Comparison of Two Learning Algorithms for Text Categorization", *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, pp. 81–93, 1994.

[18] D.D. Lewis, "Feature Selection and Feature Extraction for Text Categorization", *Proceedings of Speech and Natural Language Workshop*, pp. 212–217, 1992.

[19] Grid Computing Info Centre, http://www.gridcomputing.com/gridfaq.html

[20] GridSphere: A Portal Framework, http://www.gridsphere.org

[21] LCG-2 Middleware Overview, https://edms.cern.ch/file/498079/0.1/LCG-mw.pdf

[22] Websphinx: Personal Customizable Web Crawler, http://www.cs.cmu.edu/~rcm/websphinx/

[23] B.B. Cambazoglu and C. Aykanat, "Harbinger Machine Learning Toolkit Manual", Technical Report, BU-CE-0502, Bilkent University, Department of Computer Engineering, Ankara, 2005.