Distributed web spider based on cluster and grid technologies

Branko Dodig and Damir Krstinić, M.Sc.
Center for Scientific Computing
Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture
University of Split
R. Boškovića b.b., 21000 Split, Croatia
E-mail: branko.dodig@fesb.hr, damir.krstinic@fesb.hr
Phone: +385 21 305 617

Abstract— **We present our web spider developed as a part of a web search engine with language grammar support. Spider is optimized for distributed memory multiprocessor arhitecture, with multithreading core processors. As a part of this work, distributed data models have been developed to optimize fast manipulation and long time storage of the large amount of data generated by process of web indexing. Development of the spider is part of CRO-GRID Applications project, subproject of the CRO-GRID project.**

I. Introduction

THE purpose of the activity is the production of a web search engine with support for Croatian and other flective languages, and its applications to Data Mining[1]. Search engine is based on vector space principles where data is represented as a term-by-document matrix. The Latent Semantic Indexing technique[4] reduces the size of the matrix by exploiting higher order structure of the terms by document connections, thus improving the performance of the information retrieval system. By giving the indexing system basic knowledge of the human language, and the ability to recognize different forms of the same word, the original matrix can be reduced by an order of magnitude, and important term-document connections strengthened[2]. We are focusing on languages with complex sets of grammar rules where standard English based web indexing systems achieve low performance due to complex language structure. Before processing user queries, an index of all available documents must be generated where each record represents information about one document in the collection. This is the most computationally and memory consuming process. On vivid text collections, like the Internet, the document index must be updated periodically to reflect changes in collection content, and the quality of the complete information retrieval system strongly depends on accuracy and completeness of the document index.

In a distributed environment like Internet, documents are spread over many network resources, organized in a way that every document may point to other documents, and every document can be pointed at by many other documents, introducing multiple loops. In this case, a list of all available documents does not exist, nor could be easily acquired.

Main task of the web spider is to analyze the structure of the HTML document and to extract a list of links to other documents in the collection. Extracted links are added to the central list of available documents for later processing, and also stored together with the document for purposes of page ranking[6]. Due to the large number of available documents, creating a document index can be very time consuming. Spider performance can be improved by simultaneously processing multiple documents. Depending on the computer architecture the spider is running on, this approach varies from a multithreading single processor application for standalone workstations to large scale cluster and grid applications. In the distributed memory paradigm, major problems arise in regard to data manipulation and storage. In an environment where many processes are simultaneously and asynchronously acquiring and processing documents from distributed network resources, strong rules for data access and storage must be implemented to ensure data consistency.

II. Web spider

DURING this research, an experimental distributed web spider has been developed and successfully tested, with an emphasis on exploiting the capabilities of cluster and especially grid systems. The task of a web spider is to download documents and parse them for links to other documents, then follow these links. The process must be repeated for every document within a domain, without processing the same document twice, so a spider must have a central list of links to documents which is accessed very frequently. One of the problems with this which had to be solved first is caused by every document having a large number of links which have to be looked up in the link list to determine wether they should be inserted, and that they must be inserted in a very short time. To avoid relying on any particular database system, this is performed with hash tables using the fast `djb2` hashing algorithm which shows good performance hashing string keys and achieves good hash value distribution across the table. In our web spider the task of downloading and parsing the document and the task of coordinating the process and maintaining data integrity are separated and handled by two different components in order to maximize

Fig. 1. Web spider architecture

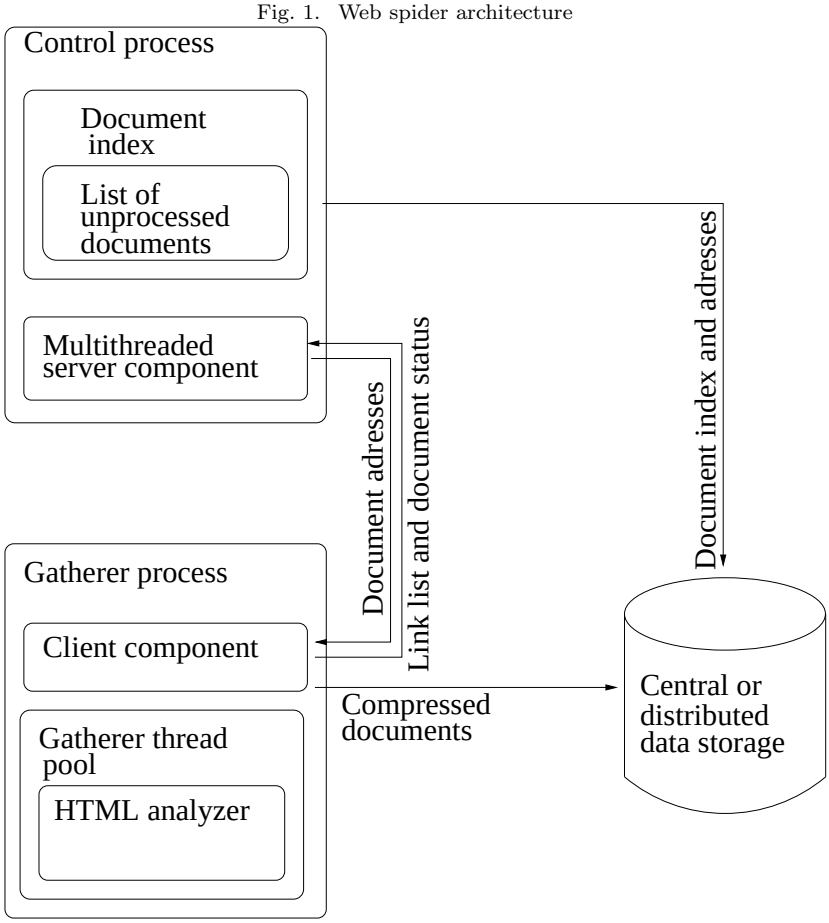

Figure 1: A schematic of web spider architecture displaying the gathering process. Each thread in the gatherer's thread pool downloads and parses a document asynchronously. The server component has the capability of serving a large amount of gatherer processes at the same time.

performance and enable parallel processing. The spider is built as a client-server system with a multithreaded server application acting as a control application which coordinates the gathering process and ensures the spider does not run in loops or processes the same document multiple times, and multiple working processes working in parallel, which download and analyze the documents for links.

Gatherer processes request addresses of unprocessed documents from the server application. After receiving a list of addresses for processing, the gatherer client component dispatches them to the threads which download and extract links from the documents. Upon completion, the links are sent back to the control application together with a request for more documents for processing. The diagram of the entire process and the architecture of the web spider can be seen on figure 1 1.

Since little can be done to reduce the time necessary to successfully download a web page as it depends on factors beyond our influence such as network conditions, server load, throughput and response times, efforts must be focused on increasing parallelism. The aim of the spider design is to fully exploit cluster and grid systems which usually have nodes with two or more multithreading processors and are capable of executing both multiple processes and multiple threads belonging to a process simultaneously. Exploiting the multiprocess and multithreaded nature of cluster and grid systems is done by executing multiple gatherer processes which themselves use multiple threads for downloading and processing documents.

Gatherer process store processed documents and the links leading from them what is necessary for page ranking either locally on the node or using a network file system such as NFS. In case NFS is used, the data is stored centrally within the cluster, achieving optimum data integrity and availability but increasing the workload on the front-end of the cluster and consumption of network bandwidth. Local storage ensures data distribution across the cluster/grid nodes but imposes a need for the control process to keep track where the data is stored for later processing. Therefore, if NFS is unavailable or undesirable, the data will be stored locally on the com-

puter and the IP address of the computer is recorded in the control process data. The two approaches can be combined, so a portion of documents can be stored centrally using NFS and a portion of documents can be stored locally. Documents are named after their document index in the control process's database, which uniquely identifies the document throughout the entire system. Document compression using the ZIP algorithm is also done, what greatly reduces collection size and enables preserving cached copies of gathered documents for web search engine purposes.

Interprocess communication is implemented using the TCP/IP protocol, what enables running on both cluster and grid systems, provided that the machine running the control process can be publicly addressed. Additionally, this approach makes it possible to utilize computer resources outside the grid/cluster systems if necessary, and is independent of the actual cluster/grid implementation. This also makes it possible to use the vast resources located in computer labs which provide greater computing power then available cluster systems and are not utilized at night nor fully used at daytime. Since interprocess communication is platform independent by nature, a Windows-based version of the gatherer application can be integrated into the existing framework. Adding the ability to operate independent of the operating system could solve the problem of exploiting Windows-based computer resources located in computer labs during daytime.

The web spider is started by running the control process which initializes the data structures and starts the server component. Gatherer processes are then started either directly or using PBS or another job management system. Gathering processes can be started or stopped at any time during the gathering process without impairing the stability or losing any data, a very useful feature in grid systems. The gathering process is considered finished when either a predefined limit or when all available documents within a domain have been harvested. Steps have been taken to solve the problem involving gathering dynamic content and the potential spider trap involved. Dynamic URL's containing session ID's are excluded, as well dynamic URL's which are longer then 127 characters to get out of spider traps faster. While this aggressive policy may reduce the amount of dynamic content gathered, it is necessary to reduce the chances of gathering duplicate content and avoid spider traps. A foolproof strategy for effectively gathering dynamic content while avoiding content duplication still remains an unsolved problem in all search engines[6].

III. Results

TESTING of the experimental cluster-based web spider was performed on a number of domains, including *klik.hr* (*www.klik.hr*) , *iskon.hr* (*www.iskon.hr*) , *blog.hr* (*www.blog.hr*), as well as a short tests on the *.hr* domain, limited to 100.000 and 150.000 sites. Measurements were done using a three node cluster with two Xeon processors per node, an internal 100 Mbps network and an external gigabit connection through a frontend computer, obtained as part of the CroGRID project. Test configuration of the web spider used one node for running the control process and a single-threaded gatherer process, and the other two nodes running a total of four gatherer processes with two threads each. Web spider operation was monitored using the Ganglia Cluster Toolkit. Network and CPU loads while processing the *klik.hr* domain can be seen on images 2 and 3. We can see that the test configuration does not exploit cluster resources fully, with overall CPU load being less then 40%. The results can be seen in table I.

Fig. 2. Network load

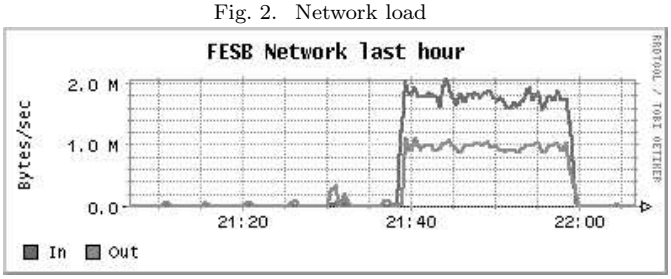

Figure 2: Monitoring network load using the Ganglia cluster toolkit while indexing the `klik.hr` domain.

Fig. 3. CPU load

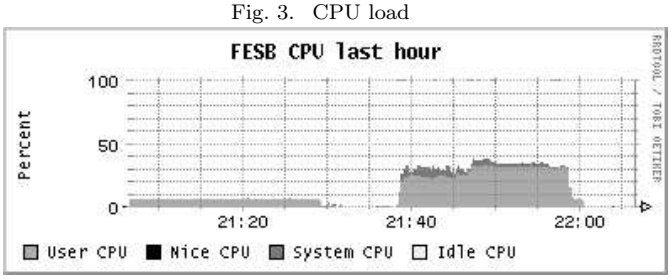

Figure 3: Monitoring CPU load using the Ganglia cluster toolkit.

Although the web spider's average speed is hard to determine due to the nature of text collections on the internet, results seen in table I show that results of around 23.000 documents per hour can be expected for this test configuration during the indexing process of the .hr domain. Significant speed degradation over time due to the increasing number of links was not observed. However, download time versus processing time ratio appears to be a very significant factor in the over-

TABLE I

INDEXING RESULTS

Gathering results for limited-domain indexing

	klik.hr	iskon.hr	blog.hr	.hr*	.hr**
documents	18902	17265	1154	100.000	150.000
time	20 min	29 min	3.5 min	4h 14min	6h 39 min
total size	1008.9 MB	839.6 MB	64.3 MB	3742.7 MB	5496.4 MB
average size	57.8 KB	54.9 KB	64.6 KB	47.08 KB	45.69 KB
success rate	94.5%	90.7%	87.1%	81.4%	82.1%
download time	50.45%	78%	87%	74.4%	76.1%
documents per hour	56762	35968	19794	23624	22556

** limited to 100.000 documents*

*** limited to 150.000 documents*

TABLE II

THREAD-DEPENDENT INDEXING RESULTS

Gathering results for the .hr domain for different numbers of threads per process

threads per process	100.000*	150.000*	200.000*	documents per hour	sucess rate
2	4h 14 min	6h 39 min	not available	22971	82.75%
10	1h 42 min	2h 27 min	3h 29 min	58951	81.44%
15	1h 40 min	2h 25 min	3h 10 min	62070	80.69%

** documents from the .hr domain*

Fig. 4. Multithreading performance

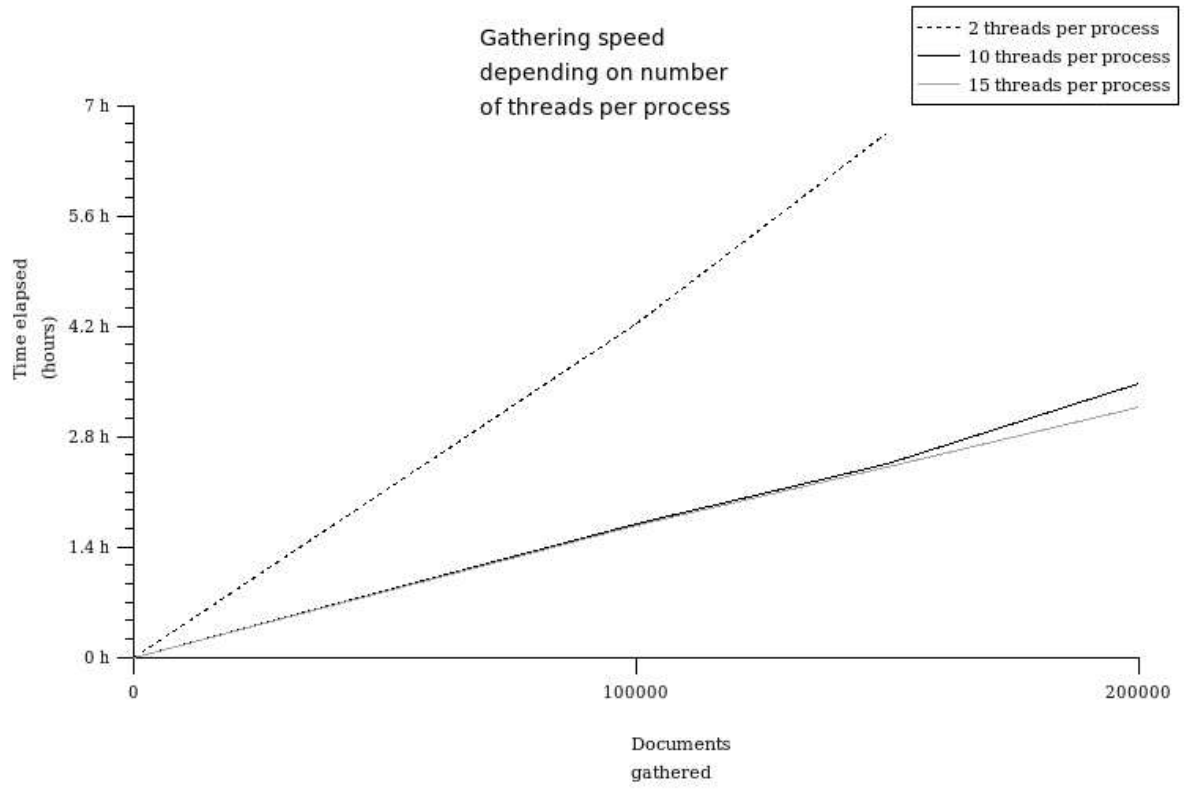

Figure 4: Gathering performance gains from increasing multithreading.

all speed of the web spider. It can be deduced from the results that the gathering speed is inversely proportional to page size and the download time vs execution time ratio. During gathering documents from very slow servers download times consumed up to almost 90% of total running time, as can be seen from gathering the `blog.hr` domain in table I. This suggests that increasing the number of threads running concurrently could improve performance by reducing the time processes wait for download to complete. Also, using distributed resources with separate network interfaces could result in a performance gain, as document download time appears to be the performance bottleneck.

To test this hypothesis, test configurations using 10 and 15 threads per process and two processes per cluster node were used to assess speed gains from increased parallelism. Results seen in table II and figure 4 show that increasing the number of threads per process from 2 to 10 yielded a 157% increase in gathering speed. Further increasing the number of threads per process up to 15 did not result in a considerable increase in spider speed, gathering documents less than 5% faster compared to the 10 thread per process configuration. During tests with both the 10 thread and the 15 thread per process configurations, CPU load on cluster nodes was between 90% and 100%, what explains the low performance gain from further increasing the number of threads. The spider speedup from gathering an average 23000 documents per hour to an average of 60000 documents per hour clearly shows the advantages of using a multithreaded spider architecture.

Success rates for .hr domain indexing are between 8% and 9% lower then the `MWP` project results from the `MWP3`[3] measurement of Croatian web space conducted between 8.9.2003 and 25.11.2003 at `SRCE`. This result is a product of the short timeout period which is set at 2.5 seconds for initiating a connection to the server, and 2.5 seconds maximum time between receiving packets before timeout occurs. Increasing timeout periods decreases spider speed significantly, but improves success rates. Measurements conducted on the *iskon.hr* domain with a timeout period of 2 seconds took 20 minutes and resulted in a success rate of 81.9% and finding 17123 documents. Table II shows that the number of threads per process does not significantly impact success rates.

IV. Conclusion

WEB spider's main purpose is to serve as a backbone for the development of an operating web search engine with grammar support for the Croatian language. To perform this task, capability of indexing large data collections using cluster and ultimately grid systems is necessary. During the last measurements of Croatian web space done as a part of the `MWP` project at `SRCE` using the *MWP Gatherer 2.0*, starting on 8.9.2003 and ending on 25.11.2003, a total of 3.194.548 HTML documents was gathered with a total size of 92.330.334.777 bytes[3]. At the present time, the `www.pogodak.hr` search engine claims to index 6.400.000 documents of various types from the `.hr` domain[7]. Our experiments have demonstrated that these documents could be gathered within a time frame of approximately four days to five days using existing infrastructures and given that web spider speed is constant. This will enable the web search engine to operate with a recent document index using existing cluster architectures. The next step in spider development is testing on large-scale grid systems gathering large text collections such as the entire .hr domain.

References

[1] Damir Krstinić, Ivan Slapničar, *Improving text search performance with grammar support* in: Workshop on information and communication technologies, SoftCOM 2004

[2] Damir Krstinić, Ivan Slapničar, *Web indexing and search with local language support*, in: Proceedings of SoftCOM 2003, pp. 488-492, Split, 2003.

[3] Measurement of Croatian web space project `http://www.srce.hr/mwp/`.

[4] Michael W. Berry, Zlatko Drmač, Elizabeth R. Jessup, *Matrices, Vector Spaces and Information Retrieval*, SIAM Review, Volume 41, Number 2, pp. 335-362, 1999.

[5] Michael W. Berry, Susan T. Dumais, Gavin W. O'Brien, *Using linear algebra for intelligent information retrieval*, SIAM Review, Volume 37, Number 4, pp. 573-595, 1995.

[6] *Google*, `http://www.google.com/technology/`

[7] *Pogodak* `http://www.pogodak.hr/`