Wojciech Turek
Edward Nawarecki
Grzegorz Dobrowolski
Tomasz Krupa
Przemyslaw Majewski

WEB PAGES CONTENT ANALYSIS USING BROWSER-BASED VOLUNTEER COMPUTING

Abstract *Existing solutions to the problem of finding valuable information on the Web suffers from several limitations like simplified query languages, out-of-date information or arbitrary results sorting. In this paper a different approach to this problem is described. It is based on the idea of distributed processing of Web pages content. To provide sufficient performance, the idea of browser-based volunteer computing is utilized, which requires the implementation of text processing algorithms in JavaScript. In this paper the architecture of Web pages content analysis system is presented, details concerning the implementation of the system and the text processing algorithms are described and test results are provided.*

Keywords volunteer computing, text processing, Web crawling

1. Introduction

The last decade brought significant growth in the amount of data available in the World Wide Web. This resulted in making the Web the greatest database of electronic data. It contains enormous amounts of information on almost every subject. According to the WorldWideWebSize.com, the most popular search services index more than 50 billion Web pages [1]. In 2008 Google published a note on its official blog, that the link processing system found 1 trillion unique addresses [2]. The amount of information available on the Internet is huge and it will be growing fast.

Popularization of the so called "Web 2.0" solutions allowed users of the Web to become authors of Web pages content. This phenomenon has many side effects, including the partial anonymization of information source. Therefore, the information published on the Web is often unreliable. Moreover, the character of the Web technology induces dynamics and lack of structure of the data.

These features create a crucial need for search systems, which would help users find valuable information. Currently, the need is satisfied mostly by the Google search service, which has over 82% of the market [3]. It turns out that the availability of information does not depend on its presence in the Web, but on the ability of finding it with the Google Search service.

All publicly available services, like Google Search, have many significant limitations. Some are deliberately created by the managing companies and some are caused by features of algorithms and technologies. The most important are:

- Simple query language, which is not suitable for expressing complex requests. For example, it is not possible to find sentences about side effects of particular drugs or find documents about a specific brand that contains profanities.
- Queries specified in a formal, constructed language, which uses particular constructions for expressing complex needs. A user must learn, how to communicate with the system, what limits its usability.
- Arbitrary sorting of results, which is based on the popularity of pages rather than compliance with a query.
- Search is based on outdated information, collected by a crawler a few days earlier. Fixed-interval indexing is sufficient only for static pages.

In this paper a different approach to the problem of finding required Web pages is described. It adopts an idea of volunteer computing for performing Web search on demand. Necessary computations are performed by Web browsers, while a user is reading a dedicated Web site. the text analysis algorithm is written in Java Script language, which can be executed by every typical browser. Provided that a sufficient number of users are involved, the approach can result in very high performance.

The basic idea of the on-demand Web searching using browser-based volunteer computing has already been presented in [4]. In this paper significant extensions concerning text processing, implementation details and tests results are presented.

In the next section an introduction to the main problems concerning Web crawling is presented. In the following section methods of volunteer computing are discussed. The fourth section presents an architecture of the real-time crawling system. The text processing algorithms are described in the fifth section, followed by tests and the results of performed experiments.

2. The problem of Web crawling

The task of Web crawling seems rather simple at first glance. A crawler must continuously download Web pages content and search for URL addresses in the content. The most typical aims of Web crawling are:

- collecting data,
- analyzing content using a particular algorithm,
- building data structures for accelerating further processing.

Most crawlers share a similar architecture, which is presented in Figure 1.

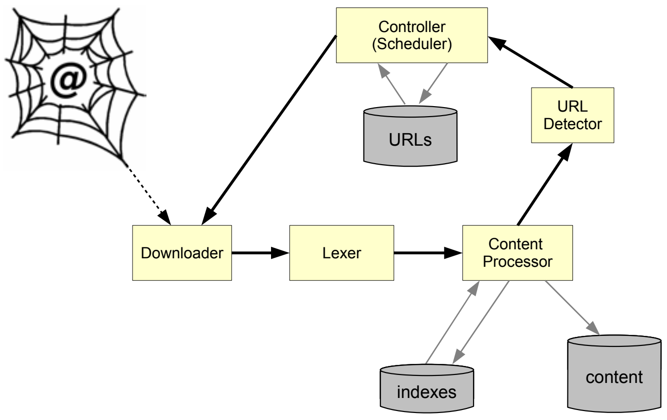

Figure 1. Abstract Web crawler architecture.

The process of crawling is managed by the Controller (or Scheduler) component. It is responsible for starting and supervising threads, which process particular Web pages. Each page, identified by an URL address is downloaded from the Web and passed to the Lexer component. After successful lexical analysis The Content Processor component builds the required indexes (for example, the index of all words found on crawled pages) and performs other specific tasks. Finally, the content is searched for URL addresses, which are passed to the Controller.

The scale of the problem is definitely the greatest challenge for the implementation of Web crawling systems. Two components of the abstract architecture presented above have to deal with scale issues: the Controller, managing URLs database and the Content Processor, responsible for updating indexes.

The Controller has to determine, if a new URL found on an analyzed Web page, should be added to the database. This is a non-trivial task, as the number of possible URLs is infinite. Moreover, the controller should avoid several types of traps existing in the Web:

- pages with many (possibly infinite) different addresses,
- generated Web pages, which contain links to other generated pages,
- link farms, used for pages positioning, etc.

The Content Processor has to create indexes of items found in the pages content. Before adding a new item, its presence in the index has to be changed. Time required for the task is dependent on the size of the index.

Another important problem, concerning the Web crawling task, is the dynamics of the content. If a crawling system works continuously, changes in the content of pages have to be reflected in indexes and content databases. Publicly available search services typically discard older versions of content, losing information which was removed from the Web.

Since Sergey Brin and Lawrence Page published the paper about their revolutionary indexing system [5], many indexing systems have been developed. Currently, several mature solutions are available, providing slightly different functionalities and performance. Some examples of popular systems are:

- WebSPHINX [6], which can use user-defined processing algorithms and provides graphical interface for crawling process visualization.
- Nutch, which can be easily integrated with Lucene [7], a text indexing system.
- Heritrix [8], used by The Internet Archive digital library, which stores Web page changes over recent years.

All existing solutions are dedicated for creating centralized crawling systems, which slowly traverse the Web and build required indexes. This approach is hardly applicable for an on-demand search with sophisticated algorithms for content analysis.

An approach presented in this paper uses many computers of Web users to perform fast processing of Web pages. Three components of the abstract architecture, the Lexer, the Content Processor and the URL Detector are executed by the user's browser. The centralized part of the system is responsible for scheduling tasks for connected browsers, sending pages and collecting results. The performance of the approach depends mostly on the number of users connected to the service.

The solution is inadequate for building complete indexes of words on all pages, as parsed content is not processed centrally. However, it can be used for building summarization of analyzed pages by extracting metadata from websites (for example by extracting the main topics from analyzed texts) and sending only these metadata to the server. Moreover, it can perform an on-demand search of current versions of Web pages very fast, using complex text analysis algorithms, such as:

- information retrieval – linguistic rules could be build to find only some specific information (for example side effect: headache caused by a specific drug),

- chunking the text into clauses to allow for better semantic analysis,
- tagging the text with part of speech information,
- providing domain-free shallow semantic analysis to allow use these of data for further text analytics applications.

This approach can provide a user with much better results, which could not be achieved using index-based search.

3. Volunteer computing using web browsers

Volunteer computing [9] is a widely known model of computation distribution among a large number of computers. The idea of the distribution is to use the computation power of a large number of nodes to significantly decrease the time of calculation. The computational power of one node is lent by the owners through a specific software program intended for a special application. The aim of the software application is to get a computational task from a distribution system, process it and return the results.

In distributed computing, two kinds of computational power are possible to lend:

- CPU time,
- GPU time.

The most renown project using the volunteer computing idea is Seti@Home [11]. The beginning of the volunteer computing projects was not easy. The first software applications were designed especially to particular calculation problems and projects. This is an inefficient approach due to the losing of the already recruited volunteers. To take part in a different computational project a user had to install a different application for each of them.

Nowadays, the approach changed and focuses on the development of a middleware layer. The aim of the middleware design is to distribute jobs and gather results with complete separation from concrete projects. This kind of new approach allows us to create one application for all the users which is also useful for the recruitment of the new volunteers.

The most popular open access framework is BOINC [10]. The architecture of the BOINC [10] framework was designed in such a way that each interested party is able to start their own research calculation project within the BOINC [10] architecture. The projects run under this framework are usually non-profit and access free. After 2008, the possibility of using GPU power for task processing appeared. The model of BOINC [10] computing is based on credits. The server side is responsible for credit allocation among the clients along with the jobs. The client side is responsible for the client-server communication, task processing and returning results. The credit feature is one of the most important parts of the system. One of the pros of credits is, that it helps to get rid of the problems with data consistency. Data is distributed to at least two volunteers and after a comparison, when data is returned, a credit is given in proportion to the data consistency. The amount of given credits is also proportional to GPU time used and then it is used for statistics.

Another well-known framework is XtremWeb [12] research and development project. It was designed to create light and flexible distributed computing networks locally on the universities, companies or any other local networks. The aim of the project was to explore the possibilities of the network and distribute computing as well as to test the peer-to-peer application in the distributed computing. The architecture of the system consists of three main parts: a client, a coordination server and a worker. The client is the entity responsible for the project definition and the worker asks the server layer for new tasks to process.

Xgrid [13] is a R&D project started by Apple in 2004. The architecture design is divided into client (project initialization, task sending), controller (data partition, sending to different agents, results assembling and return to the client) and agents (task processing, result returns to the controller). In the Xgrid project the BEEP protocol is used for communication purposes.

The last but not least project which is worth mentioning is Grid MP [14]. This commercial product was designed with a lot of features, such as a task scheduler with priorities, volunteer monitoring, security controls. The architecture of the systems consists of a service interface, a management console and an agent. The service interface (MGSI WebAPI) provides the ability to define services by developers among the whole system. The management console provides the ability to monitor, control and manage all the system. The MP agent is a client software which is responsible for task completion and results returning.

The Web browser-based volunteer computing is not a brand new idea for distributed computing. The very first project, already mentioned in this paper was, Bayanihan [9]. It was based on a Java applet as the client-software application. When the volunteer accessed a Web page with this applet embedded the process started. The client communicated with a server asking for a job task, processing it and returning the results.

Anothe project called Gears [15] is also a framework based on the BOINC. It is based on JavaScript client software and the BOINC middleware architecture. A similar project has been started by Siggi Simonarson [16].

Summing up, the use of Web based distributed computing is not a brand new idea as it already was explored a bit in previous projects.

The main advantage of using JavaScript based distributed computing is its implementation in every Web browser on the market. Taking the advantages of this into account, the main potential of recruitment of a large amount of volunteers is possible to explore. Other reason for using the JavaScript technology is its performance, which has grown significantly over the last few years.

A large part of the features of this language has been implemented in the same way within many Web browsers so it is portable. This feature can make anybody with a netbook, notebook, mobile device etc. a volunteer – this provides the possibility of having a lot of users recruited. JavaScript is also very easy to use, so the process of prototype development is very rapid. Nowadays, it also uses system resources in a much more efficient manner.

On the other hand, there are slight implementation differences in different browsers. Other disadvantage is the user interface that is blocked in the meantime of the program execution which is not impossible to overcome but has to be implemented additionally. Also, to achieve maximum performance, usage of HTML5 is recommended. Unfortunately it is incompatible with older browsers.

Recently, along with HTML5 and WebGL implementation JavaScript developers gained an access to GPU. It might be used also as a computational power provider.

Web browser plug-ins are a slightly different approach. It is embedded as a part of the browser, not a Web page, so it gives us a bit better performance than a JavaScript within a Web page. To the advantages of this approach we could add the ability to use operating system functions encapsulated by the Web browser API. The second advantage is the thread based execution of the code which does not interrupt/interact with the user interface so automatically it won't disturb the user anyhow. Unfortunately, on the other hand, a plug-in must be re-written for every different browser and different version of a given browser. This is the main disadvantage of the plug-ing. It might also not be compatible with newer versions of Web browser. Moreover, it creates a limitation in the volunteers recruitment process due to the fact that plug-in has to be downloaded and installed by the user within a browser.

In this paper we present JavaScript based approach to the volunteer computing. We use the advantages of this technology and present a different approach to the idea of Web crawler computation.

4. Architecture of the system

The Web Content Analysis System, which is presented in this paper, aims at performing complex text processing for finding Web pages meeting specified criteria. The computations must be parallelized because of the complexity and huge amounts of data to be processed. Details on the text processing algorithm will be presented in the next section.

The system will adopt the paradigm of volunteer computing to gain required computational power. However, the volunteers will not be forced to install any dedicated software. Instead their Web Browser will be used – the computations will be performed in the background, while a user browses prepared Web pages. The background computations will be performed by dedicated components implemented in JavaScript.

A user of the system will define particular search tasks by providing the configuration of the text processing algorithm and the specification of Web pages to process. This requires at least one URL address and information on crawling policy – how many links to follow.

The general architecture of the system is presented in Figure 2.

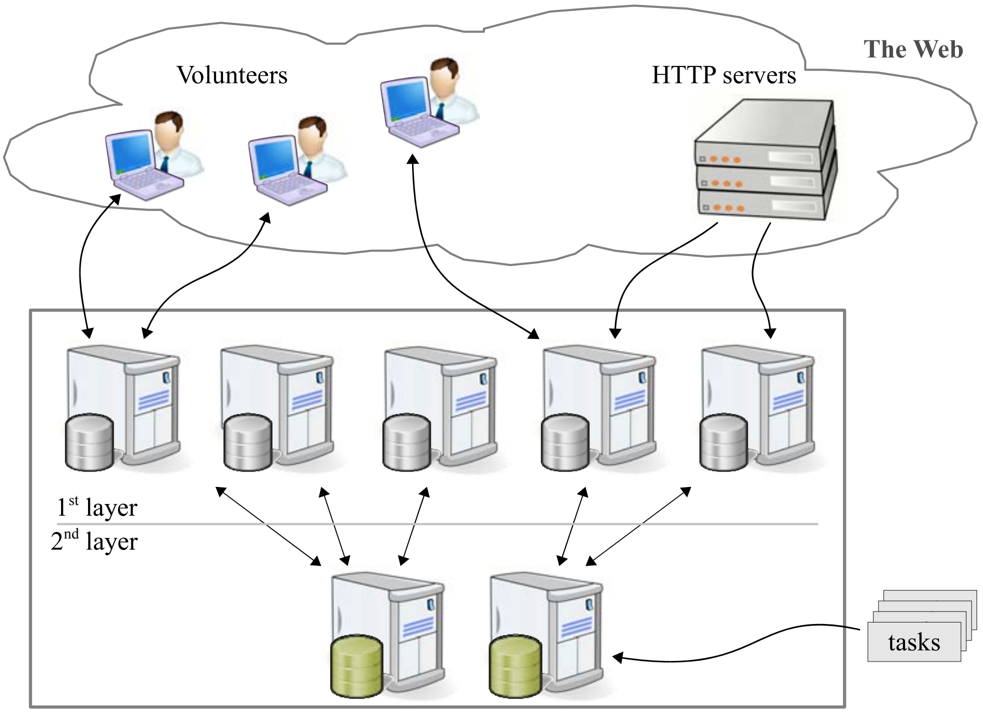

Figure 2. General architecture of the Web content analysis system.

The system is divided into two layers. The first layer is responsible for communication with volunteers, the second layer manages servers in the first layer, distributes tasks and provides results to the user of the system.

Tasks created by a user of the system are initially processed by a second layer server. It generates a proper processing scripts and sends these to the managed servers of the first layer. During the processing, the server receives the results from the managed first layer servers. The results contain information about pages meeting search criteria and about found URL addresses. The server is responsible for distributing the URL addresses among assigned first layer servers, using a load-balancing strategy.

The main responsibilities of a first layer server are shown in Figure 3.

Each first layer server has a local URL database which is extended with new URLs by the managing second layer server. There are two main tasks performed by the server simultaneously:

1. Downloading the content of Web pages.
2. Sending content to volunteers and collecting processing results.

The first responsibility aims at filling a queue of Web content to be analyzed. Downloading is performed in parallel by a specified number of threads. The size of

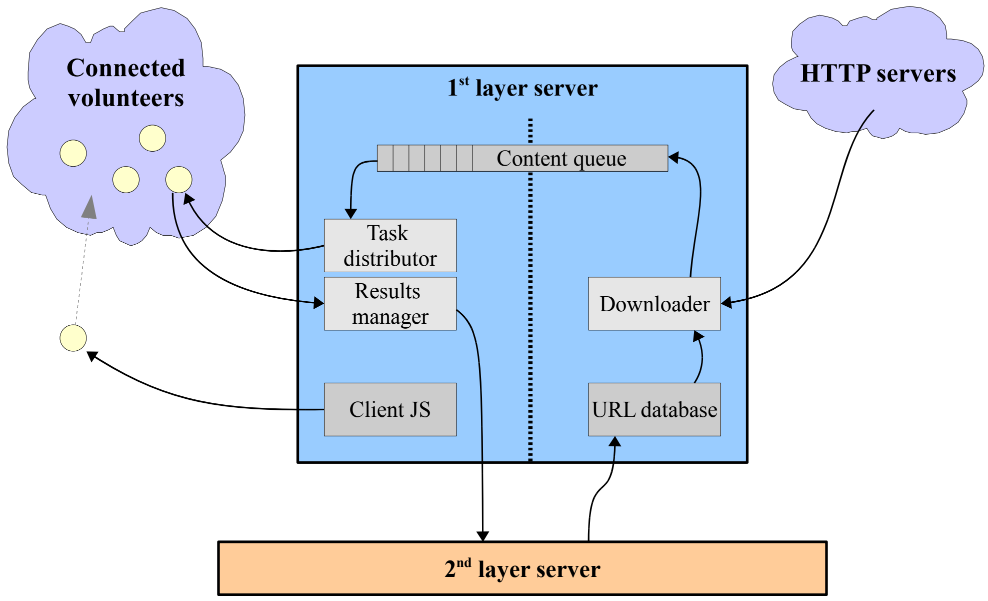

Figure 3. Processing performed by a first layer server.

the queue is configurable – typically it should contain several hundreds of web pages for smooth processing operation.

It would be much more efficient to move the downloading responsibility to a volunteer. It could download a specified Web page, perform the processing and return results to the server. Such a solution would reduce network traffic as well as first layer servers load. However, it is not possible to implement this behavior in JavaScript executed by a Web browser. This is caused by security reasons – one page cannot access the content of a different page downloaded from a different Web server.

A newly connected volunteer receives a prepared script with definition of search rules and an implementation of the text processing algorithm. Then it starts querying for new tasks. Each task is a single Web page content, previously downloaded from the Web by the server. The content is processed using a provided algorithm and the results are returned. The results contain information about meeting search criteria and found URL addresses. All results are returned to the managing second layer server.

The implementation of the algorithm executed by each volunteer requires HTML 5.0 JavaScript. This creates an important requirement: a volunteer must use a relatively modern browser. The reason for choosing this technology is the fact that it provides a mechanism for Web workers and gives a promising opportunity for GPU usage. The mechanism for Web workers allows for performing calculations without influencing the operations performed on the website.

The servers applications are written in PHP. The system has a dedicated API by which adding a new kind of server in a different language is possible. In the same way there is wide range of possibilities to add new types of JavaScript clients which can perform different tasks.

The architecture presented in this section was successfully tested in a real environment. It has revealed several important advantages. One of it was that the calculation has been pushed to the users side. It has significant impact on the computational power which was used for text analysis algorithms. Other advantage is that it provides very flexible and elastic methods for scaling the system.

In the following sections the details on the text processing algorithms are described and the tests results are presented.

5. Advanced text processing

The text processing algorithm, designed for classifying text, is based on a set of linguistic rules. A rule is a set of conditions; a particular text must meet these conditions in order to **match** the rule.

At the beginning, the text is being split into words and sentences (lists of words). All whitespaces and punctuation marks are removed from the text. Then, every sentence is tested against every rule. If any of the rules matches any of the sentences, the text matches the classifier defined by the set of rules.

The rules are defined in a formal language, as they are processed automatically by the algorithm. The language must be able to express complex conditions concerning the text. Also it has to be human-readable, as in current version of the system the rules are defined manually. The language was designed for matching an inflectional language and tested on Polish examples.

A rule of a language can be:

- a string, which will match only and only a single word containing the same list of characters as the string,
- one or two rules joined with an operator; available operators are listed and described in Table 1.

Additionally, each rule can specify a modifier for additional requirements concerning the length of the matching sentences. The requirements are listed and described in Table 2.

For a shorter notation, each rule can have a name assigned. The name can be used instead of the rule in the next rules. A simple example of three rules in Polish is presented below.

M_wojna:

```
    wojna[/]wojny[/]wojnie[/]wojne[/]wojnom[/]wojnami[/]wojen[/]wojnach[/]
    krucjata[/]krucjaty[/]krucjacie[/]krucjate[/]krucjat[/]krucjatom[/]
    krucjatami[/]krucjatach[/]bitwa[/]bitwy[/]bitwie[/]bitwe[/]bitew[/]
```

`bitwom[/]bitwami[/]bitwach[/]inwazja[/]inwazji[/]inwazje[/]inwazjom[/] inwazjami[/]inwazjach`

M_pomagac:

`pomoc[/]pomagac[/]pomagam[/]pomagasz[/]pomaga[/]pomagamy[/]pomagacie[/] pomagaja[/]pomocny[/]pomocna[/]pomocne[/]pomoge[/]pomozesz[/]pomoze[/] pomozemy[/]pomozecie[/]pomoga[/]pomoglem[/]pomoglam[/]pomogles[/] pomoglas[/]pomogl[/]pomogla[/]pomoglo[/]pomoglismy[/]pomoglysmy[/] pomogliscie[/]pomoglyscie[/]pomogli[/]pomogly[/]pomoz[/]pomozcie[/] pomagaj[/]pomagajcie`

(M_wojna[+<5]M_pomagac)

Table 1
Operators for defining linguistic rules. Priority can be modified using brackets.

Operator	Arity	Priority	Description	Example
—	2	1	optional space	un-friendly
*	1	1	zero or more characters	responsib*
!	1	1	NOT – negation of a rule	!play
[/]	2	5	OR – any of two rules in the sentence	see[/]saw
[&]	2	4	AND – both rules in the sentence	set[&]bomb
[+]	2	3	successive rules in the sentence	dirty[+]bomb
[+=N]	2	3	successive rules in the sentence separated with N words	build[+=2]bomb
[+>N]	2	3	successive rules in the sentence separated with more than N words	buy[+<2]bomb
[+<N]	2	3	successive rules in the sentence separated with less than N words	components[+>3]bomb
[&=N]	2	2	both rules in the sentence separated with N words	hit[&=2]stone
[&>N]	2	2	both rules in the sentence separated with more than N words	profess*[&>4]assassin*
[&<N]	2	2	both rules in the sentence separated with less than N words	enjoy[&<3]party

The first rule detects the presence of a single word related to warfare or battles. The second matches different forms of words related to helping. The last rule, which uses previously defined ones, detects sentences related to the glorification of war.

The language allows for defining complex and accurate rules. Defining high quality rules (simple, yet accurate) is rather a hard task, which is currently performed manually by linguistic experts. Tests on different domains showed that accuracy of 90–97% can be achieved this way.

Automated generation of rules has also been considered; methods of machine learning could be utilized for that. This is definitely one of the most interesting directions for the further development of the algorithm.

The algorithm for matching sentences against the rules is implemented using a regexp engine. The set of rules is translated into a JavaScript file which can be executed in a browser. The result of script execution is the decision whether the text matches the set of rules or not.

Table 2
Modifiers for specifying length of sentences.

Modifier	**Description**
{_userwordcount<N}	sentence has less than N words
{_userwordcount==N}	sentence has N words
{_userwordcount>N}	sentence has more than N words
{_usercharcount<N}	sentence has less than N characters
{_usercharcount==N}	sentence has N characters
{_usercharcount>N}	sentence has more than N characters

6. Tests and results

The system presented in this paper has been implemented and carefully tested. The tests aimed at verifying its usability and measuring its performance.

In real applications the system makes use of computers of the Internet users, which will provide different computational power and will abandon the platform on a random basis. This kind of experiment would not provide any comparable results. The tests were designed to measure the performance of particular aspects of the system. This required using known and controllable environment, like a cluster of identical computers.

Two parts of the experimental system – the servers and the clients – were geo-distributed and connected with a fast public Internet network. Up to 10 servers have been used, up to 9 of those served as first layer servers, while the second layer consisted of one server. The client side used up to 20 physical computers equipped with 4 core CPU and 4 GB of RAM. Each client computer was running 10 web browsers simultaneously. Use of several Web browsers on a single computer is justified because it improves the utilization of multi core CPUs. This means that the system controlled up to 200 volunteers simultaneously.

Two sets of tests have been performed:

1. Tests of the text processing algorithm.
2. Tests of the Web content downloading and processing.

The tests of the text processing algorithms were based on a set of 3000 plain text samples of similar length (several thousand words each). The samples were located in server database and loaded into source queue when needed. This means, that in these tests there was no Web pages downloading at all. The text processing algorithm was configured for finding extremist right-wing content.

The tests showed, that the use of several first layer servers in this scenario was pointless. 200 clients were served by a single server without any delays, therefore parallelization of tasks distribution did not influence performance. The results of the experiments for a different number of client computers are shown in Figure 4.

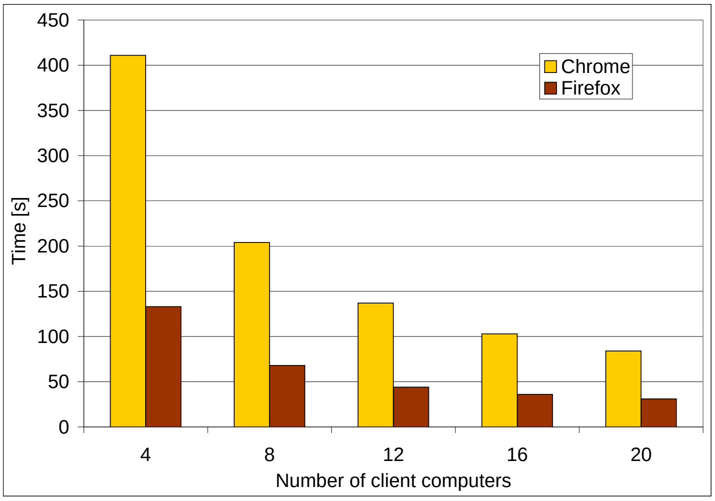

Figure 4. The results of the distributed text processing using a different number of client computers.

The first striking result of this test is the difference in performance between two Web browsers used. Firefox (version 14) performed 3.2 times better that Chrome (version 21). This shows a very significant difference in the implementation of the JavaScript interpreter used by the browsers. Other browsers have not been tested because required features of HTML 5 have not been supported.

The system showed very good (almost linear) scalability in this test case. This result was expected as the tasks were independent and relatively long-lasting – each took between 2 and 8 seconds. The server was almost idle after the first seconds, when initial data was sent to all clients.

The second set of tests aimed at testing the integrated Web pages search system. The same text processing algorithm was used, however, the text for processing was downloaded by a crawler from the Web. The initial set of URL addresses was set for several popular news portals.

Each test lasted 180 seconds. Client computers used Firefox browsers. Figure 5 shows the results of the experiment for different number of client computers and a different number of first layer servers.

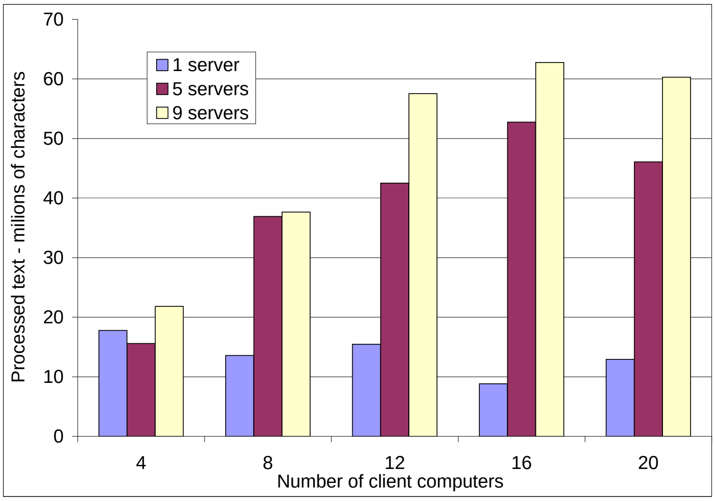

Figure 5. The results of the integrated Web pages search system tests.

The results show that a single first layer server is sufficient for 40 clients and is insufficient for 80 clients. The use of 9 servers in the first layer was justified when the number of clients exceeded 80. A further increase of the computational power provided by clients did not give expected performance gain. It was influenced by the network throughput and HTTP servers performance, which reached its limits at about 50–60 millions of characters during the test period.

The system proved its usability and stability in this test scenario. Large amounts of data have been processed by advanced text analysis algorithms using parallel, browser based approach.

7. Conclusions

The solution described in this paper aims at solving several issues, which reduce the usability of popular tools for finding information on the Web. Most significant limitations of existing solutions include inconvenient query languages, arbitrary sorting or results and out-of-date information.

The Web pages analysis system described in this paper can provide up-to-date results, which are available on the Web at the moment of searching. Advanced text processing algorithms implemented in JavaScript are executed on volunteers computers, making it possible to parallelize the processing and significantly increase processing speed.

Further tests of the systems will include real-life scenarios involving Internet users as volunteers.

Acknowledgements

The research leading to these results has received funding from the Polish Grant (funded by The Ministry of Science and Higher Education) with agreement number 0108/R/T00/2010/11.

References

[1] Kunder M.: *WorldWideWebSize.com*, 20.09.2012

[2] Alpert J., Hajaj N.: *We knew the web was big…*, `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`, 25.07.2008

[3] Net Applications.com, *Search Engine Market Share*, `http://marketshare.hitslink.com/search-engine-market-share.aspx`, 20.09.2012

[4] Krupa T., Majewski P., Kowalczyk B., Turek W.: On-Demand Web Search Using Browser-Based Volunteer Computing. *Proc. of Sixth International Conference on Complex, Intelligent and Software Intensive Systems*, pp. 184–190, Palermo, Italy, 2012.

[5] Brin S., Page L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Seventh International World-Wide Web Conference*, Brisbane, Australia, 1998

[6] Miller R. C., Bharat K.: *SPHINX*: A Framework for Creating Personal, Site-Specific Web Crawlers. *Proc. of WWW7*, Brisbane Australia, 1998.

[7] Shoberg J.: *Building Search Applications with Lucine and Nutch.* ISBN: 978-1590596876, APress 2006.

[8] Sigursson K.: Incremental crawling with Heritrix. *Proc. of the 5th International Web Archiving Workshop*, 2005.

[9] Sarmenta L. F. G., Hirano S.: *Bayanihan: Building and Studying Volunteer Computing Systems Using Java.* Future Generation Computer Systems Special Issue on Metacomputing, vol. 15, no. 5/6. Elsevier Publ., 1999.

[10] Anderson D. P.: BOINC: A rb1 System for Public-Resource Computing and Storage. *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[11] Korpela E., Werthimer D., Anderson D., Cobb J., Leboisky M.: SETI@home-massively distributed computing for SETI. *Computing in Science & Engineering*, 3(1): 78–83, 2001.

[12] Cappello F., Djilali S., Fedak G., Herault T., Magniette F., Néri V., Lodygensky O.: Computing on large-scale distributed systems: XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21 (3): 417–437, 2005.

[13] Buyya R., Ma T., Safavi-Naini R., Steketee C., Susilo R.: Building computational grids with apple's Xgrid middleware. *Proc. of Australasian workshops on Grid computing and e-research*, pp. 47–54, 2006.

[14] Venkat J.: Grid computing in the enterprise with the UD MetaProcessor. *Peer-to-Peer Computing. Proc. Second International Conference* on. 2002.

[15] Gears: *Gears project*, `http://webcomputing.iit.bme.hu/`, 4.12.2011

[16] Simonarson S.: *Browser Based Distributed Computing*, TJHSST Senior Research Project Computer Systems Lab. 2010.

Affiliations

Wojciech Turek
AGH University of Science and Technology, Krakow, Poland, `wojciech.turek@agh.edu.pl`

Edward Nawarecki
AGH University of Science and Technology, Krakow, Poland, `nawar@agh.edu.pl`

Grzegorz Dobrowolski
AGH University of Science and Technology, Krakow, Poland, `grzela@agh.edu.pl`

Tomasz Krupa
Fido Intelligence, Gdansk, Poland, `tkrupa@fidointelligence.com`

Przemyslaw Majewski
Fido Intelligence, Gdansk, Poland, `pmajewski@fidointelligence.com`

Received: 22.10.2012
Revised: 17.01.2013
Accepted: 11.02.2013