A web crawler design for data mining

Abstract

The content of the web has increasingly become a focus for academic research. Computer programs are needed in order to conduct any large-scale processing of web pages, requiring the use of a web crawler at some stage in order to fetch the pages to be analyzed. The processing of the text of web pages in order to extract information can be expensive in terms of processor time. Consequently a distributed design is proposed in order to effectively use idle computing resources and to help information scientists avoid the need to employ dedicated equipment. A system developed using the model is examined and the advantages and limitations of the approach are discussed.

1. Introduction

The economic and cultural importance of the web has guaranteed considerable academic interest in it, not only for affiliated technologies, but also for its content. Research into web pages themselves has been motivated by attempts to provide improved information retrieval tools such as search engines, but also by the desire to know what is available, how it is structured and to determine its relationship with other meaningful human activities. One strand has been the counting of links between academic web sites in order to construct an analogy to the journal Impact Factor. The figures required for this calculation can be obtained using the advanced facilities available in search engines such as AltaVista and Infoseek, but their use has raised questions of reliability that have led to the creation of a specialist web spider/analyser to produce the raw data by direct crawling and analysis of the sites concerned. Information scientists and others wishing to perform data mining on large numbers of web pages will require the services of a web crawler or web-crawler-based tool, either individually or collaboratively. The potential power of web mining is illustrated by one study that used a computationally expensive technique in order to extract patterns from the web and was powerful enough to find information in individual web pages that the authors would not have been search engine Google, which uses mathematical calculations on a huge matrix in order to extract meaning from the link structure of the web. The development of an effective paradigm for a web mining crawler is, therefore, a task of some importance.

A web crawler, robot or spider is a program or suite of programs that is capable of iteratively and automatically downloading web pages, extracting URLs from their HTML and fetching them. A web crawler, for example, could be fed with the home page of a site and left to download the rest of it. A sophisticated web crawler may also perform additional calculations during the crawl in order to identify pages judged relevant to the crawl or to reject pages as duplicates of ones previously visited. One important role for crawlers is to support the action of search engines, normally assisted by other programs that construct the searchable index after the crawler has obtained the pages. If data mining is required, then it is possible to either build this into the crawler or into a separate program. In the normal course of operation, a simple crawler will spend most of its time awaiting data from remote computers as part of the process of requesting and receiving a web page. For this reason, crawlers are normally multi-threaded, so that hundreds of web pages may be requested simultaneously by one process. If, however, the crawling task requires more complex processing, then clearly the maximum number of threads that can be efficient handled will be reduced, and consequently the speed of the crawler too. This paper presents a distributed approach for crawlers including computationally expensive additional functionality, such as data mining. It works by operating as a distributed system, with a central control unit allocating tasks to idle computers connected to a network.

Distributed systems are not new. The idea of using idle computers connected to the internet or other network as a means of gaining extra processing power has been used many times before. A large-scale distributed system for web crawling has also been proposed, but one aimed at general web searching rather than for specific tasks. Large search engines can also widely distribute processing power, including Google, which uses “thousands of PCs linked together”. Much smaller-scale crawlers have also been is use for some time, for example personal site-specific spiders. These run on single personal computers and their performance, in terms of the number of pages that can be processed in any given time period, is limited by that fact. For applications requiring extensive processing by the crawler, a single personal computer solution may be fast enough. An alternative is a distributed model in which a central control unit allocates tasks to many crawlers operating on individual personal computers. This is a practical solution in any organization in which many computers are left unused for periods of time, for example weekends. Many universities would also make ideal candidates, with whole laboratories of computers left idle during holiday periods. An architecture for such a system will be discussed, one that is easy to create, install and run, even through firewalls.

2. Architecture

The proposed design must be capable of operating in a networked environment with minimal disruption to its normal functioning. To this end, four constraints were placed upon the implementation of the general concept.

1. Almost all processing should be conducted on idle computers.
2. The distributed architecture should not significantly increase network traffic.
3. The system must be able to operate through a firewall.
4. The components must be easy to install and remove.

2.1 The crawler/analyser units

The crawler is the program that will crawl a site or set of sites, analyse the pages fetched and then report its results. It will need to be written so that it can execute on the type of computers on which there will be spare time, normally personal computers running a version of Windows, and so could be written in C, Java, Visual Basic or Delphi, for example. The crawler will need to access permanent storage space to save the web pages downloaded before analysing them. This can be achieved by linking to a database, if the target computers are known to all have a common one capable of handling enough data. In other situations a simple solution is to use the normal file storage system, saving the pages as individual files. In either case, in order to make the program easily removable, it must provide a facility for the user to delete all saved data. Pages must be saved on each host computer, rather than transmitted to the control unit, in order to minimize network traffic. This precludes the use of a large-scale sever-based database to store web pages. The interface must include an immediate stop button visible when it is running, and a button to clear all data from the computer. It is also recommended to include a button to stop the crawler once it has completed the current task. This is useful for cases when it is known in advance that a computer, or set of computers, will be needed for other purposes at a given time in the future.

2.2 The control unit

The control unit will live on a web server and will be triggered by it when a crawler unit requests a job or sends some data. The ability to start a program in response to a particular web page request is standard to the common web servers such as Apache and Internet Information Server as part of their implementation of the Common Gateway Interface. The program can be written in any language capable of running on the server computer, with common choices for CGI applications being Perl, ASP, C and Python. It will need to store the commands that the owner wishes to be executed, together with their status, whether they have been completed, have been sent to a crawler, or are as yet unallocated. A simple way of implementing this is as a text file saved on the server. In order to make the system robust in an environment where individual components can be stopped without warning, a feature must be included to automatically identify incomplete jobs, and then to reallocate them either with or without human intervention. This allows, for example, the owner of the computer on which any crawler unit is operating to stop it without letting it complete its current job.

2.3 The messaging system

In order to satisfy the first and second requirement above, the crawler units must be almost completely self-contained, and able to download and process entire websites with little or no communication with the control unit. This can be achieved by reducing the communication between the two to that concerning the nature of the task to be executed, results reporting and status checks. Interaction is therefore limited to the following messages.

- the crawler unit, upon starting or completing a crawl, sending a message to the control unit indicating that it is ready to execute a new request;
- the control unit sending an instruction to the crawler unit indicating a site to crawl, and the type of processing to be performed on the downloaded site;
- the crawler unit reporting its results once the crawl has been completed.

This will only work, however, if the crawl result to be reported are significantly smaller in size than the web site. If, for example, the entire uncompressed web site were to be 'reported', then this would represent an unacceptable doubling of network use.

Firewalls can cause problems to distributed systems because they are designed to stop access to specified information and resources to all those with computers running outside the protected network. A firewall, for example, may protect a web server by only allowing computers outside of a given network to send information into it that can be identified as part of the process of requesting and downloading a web page. The distributed crawler design described here can operate through this kind of firewall by hosting the control unit on a web server and coding all communications between the crawler and control units as web page requests to the control unit and web pages returned. A firewall can also be configured to stop all computers outside a network from requesting information from the computers inside it, whilst allowing those inside to request any external information. As an example of this, networks of open access student computers may be configured so that the computers can use all the resources of the internet, but outside users would be prohibited from sending in any unrequested information. The distributed design circumvents this problem by coding information sent to the crawler units from the control unit as web pages.
The particular problem that cannot be avoided is that a control unit outside the network could not initiate communication with computers inside, but only send information in response to a request for it.

In summary, the architecture will work for any system where the control unit is on a public access web server and the client units are on computers with permission to access the web. This is possible because all the information communicated is encoded in web page requests and responses. It does not allow individual crawler units to communicate with each other, nor the control unit to initiate communication with any crawler, but this, in fact, is not a problem.

3. A distributed web analyser

The architecture described above was employed to create a system for the task of analysing the link structure of university web sites. The need for this was rooted in previous experience in running a single crawler/analyse program. This program had not run quickly enough to cover the necessary number of web sites, and so it had been individually set up and run on a number of computers simultaneously. The inefficiency of this approach, in terms of both human time and processor use, led to the specification of the distributed system as a logical progression from the existing technology. In this new system, the existing standalone crawler was used as the basis for the new crawler unit component, greatly saving development time. The main modifications made were in adding the communication mechanisms and features, allowing it to be easily installed and removed. The latter changes resulted in the new program being available at a single file small enough to fit on a floppy disk and having buttons to instantly close the program and remove any saved data. The program was written in a rapid application development language in order to be able to quickly design and modify components of the analyser. It was then processed by an executable compressor in order to get it and all supporting library files to fit on one disk as a single file.

One feature that was built into the crawler component was a choice of the types of checking for duplicate pages to be used in a web site crawl. There were there options:

- use no page checking - assume that two pages with different URLs are different pages;
- the HTML page checking - reject any new page (except a frameset page) which has identical HTML to a previously retrieved page;
- use weak HTML page checking - reject any new page (except a frameset page) which has at least (say) 95% identical HTML to a previously retrieved page. This option was designed to pick up pages which contained a text counter, or similar incidental changeable page component, so that if the same page is downloaded from two different URLs, the HTML would be slightly different the second time.

The inclusion of three different crawling methods was necessary, because one of the tasks to be given to the system was comparing the output produced with the different versions, and other tasks to be processed later would require very precise results. This activity did, in fact, add considerably to the workload of the program for the larger sites. In order to minimize the work, instead of comparing each page against all of the others, various numbers were calculated from the text of each page, and only if enough of the numbers matched between two pages were they fully compared to see if the new one was to be rejected as the same, or essentially the same, as the other one. A version of this technique has been used in another search engine, Mercator, and may well be used by others. The simplest of the numbers was the length of the page, with the others calculated from the page content. This technique speeded up the process, essentially because the derived numbers were kept in main memory, and so could be checked very quickly, whereas comparing the actual pages would require them to be fetched from the hard dist first, slowing down the process considerably.

The analyser component inherited most of the analytical capabilities of its predecessor, in addition to some new ones. The three main types of tasks that it could be commanded to do were as follows.

- summarise the page and link structure of a given web site;
- process individual pages or framesets to extract information such as meta tag, Java and JavaScript use;
- use page-scraping techniques in order to report on the presence or absence of web sites in as set of search engines

In addition to the main analyses, there was a default 'idle' task which was sent when all control unit commands had been allocated. This instigated a half hour wait before repeating the request for a new instruction. This was essential to the smooth running of the system.

The control unit was entirely new and consisted of a simple Perl program. In addition to the functionality described in the previous section, it was given a reporting facility so that it could deliver to the owner a summary of which computers had crawler units allocated to tasks and which tasks were completed, allocated and unallocated. This provided a useful mechanism for checking that all was proceeding as expected, and allowed manual identification and checking for computers that had been switched off or which were malfunctioning.