Snaking the Web

EuroPython 2006 Conference

Written by Markus Franz

(mail@markus-franz.de)

ABSTRACT

Two research projects of NEC and the University of Hannover could not be realized because of technology faults. With Python, an 18 year old student from Germany solved the problems and created a new-class metasearch engine for the WWW. The development process with problems like high-volumes of data, web crawling, performance issues and other topics was easy with Python like with no other programming language.

This paper covers my experience with Python as the development language for the search engine Metager2 (www.metager2.de) and

INTRODUCTION

First let's keep in mind what a metasearch engine generally does: It is a special type of search engine that sends user requests to several other search engines and/or databases and returns the results from each one. They allow users to enter their search criteria only one time and access several search engines simultaneously. Since it is hard to catalog the entire web, the idea is that by searching multiple search engines you are able to search more of the web in less time and do it with only one click. The ease of use and high probability of finding the desired pages make metasearch engines popular with those who are willing to weed through the lists of irrelevant matches. Another use is to get at least some results when no result had been obtained with traditional search engines.

In the year 1996 the search engine lab at the University of Hannover started a research project called MetaGer2. Until today the leader of the lab has been Dr. Sander-Beuermann. The ideas was to enhance search results by adding a new level to the common metasearch concept: They wanted to build a metasearch engine that not only passes the search query to several other search engines and parse their result pages, but also to do a HTTP request to each webpage that was found. At that time the developers hoped to find a way to realize this idea in order to have a search engine that never shows broken-links and that can do dramatically improved spam detection because they always have the up-to-date contents of every result. Almost at the same time the NEC research laboratory started working on such a metasearch engine, they called it Inquirus.

But the development groups had the following problems:

- How can many websites be loaded fastly? Common metasearch engines load up to the first 10 result pages of each source search engine. If one tries to load the first 10 result pages with each page containing 10 search results, you get the addresses of 100 webpages – from one search engine! But to be a good metasearch engine you have to use as much source search engines as you can. So: 10 source engines, 10 result page with each one containing 10 results in 1000 HTTP request to be done to get the contents. [Unique results from InfoSpace -> study]
- The data returned by the loading process has to be handled fastly, too. If you say there are 15 KB per page, the new-class metasearch engines has to handle about 15 MB at the time of search. Parse the HTML, extract important information, do spam detection and rank the webpages.

You see: The concept of a new-class metasearch engine loading every result could even be done, but responsing to search query would take very much time. Development versions of both University of Hannover and NEC needed about 15 minutes to show results to a search request. But the time limit users give a search service they use is about 5 seconds. And if their query won't be finished within this time, the will no longer use this search engine and use the service of a competitor.

So they both gave up their research projects because they thought the idea of a new-class metasearch engine could never be realized.

RESTART OF DEVELOPMENT

I've been interested in information retrieval and search technologies since I was 14. My most successful project until 2004 was called Orase, I was developed for the German science youth competition Jugend Forscht. It was written in PHP5 and I was a case study provider for the introduction of PHP5. Orase was very trivial: Based on the search query it tried to guess relevant domains and then did HTTP requests on it. Then, in August 2004, I discovered a paper of Dr. Sander-Beuermann about MetaGer2 and NEC Inquirus.

With the loading system of Orase being in my mind, I thought about the idea of MetaGer2 and Inquirus. Then, in a school break, I called Dr. Sander-Beuermann and I told him that I think I've got the solutions for both the loading and the analyse problems. I told him that he is wrong with saying that such a new-class metasearch engine could not be realized. He said that we should talk about it – and we did.

Dr. Sander-Beuermann trusted in my knowledge and he organized support of search engine lab of the University of Hannover and the SuMa-eV for me. By the way: The SuMa-eV is a non-commercial registered association for helping search engine developers in Germany, they also support the P2P search engine YaCy.

GENERAL SOLUTION

As the general solution a simple, but effective idea came in my mind: Why do all HTTP request one after each other, why not parallelize this process? And why not parallelize the analyses of each retrieved webpage?

I’ve already played with parallelization technologies because of my work at Orase: The domains guessed from the user’s input were loaded *in parallel*. The loading script of Orase was written in Perl because dealing with processes was quite unstable in PHP at this time and was also not enabled by my webhoster. But Orase did not have to load more than 50 webpages in parallel, so I knew that I must invent something new to have an even much faster solution for retrieving up to 1000 webpages.

PROGRAMMING LANGUAGE

I wanted to create Metager2 with a completely new codebase, not using old code that was written in 1996 for the research project at the University, nor code from Orase. So I had to decide what programming language to use. The points were important for my decision:

- Support for threads, processes and asynchronous sockets. (I needed all three because I wanted to test which combination is the best way.)
- Ease of use: I planned to code the engine in the Christmas holidays that only last for 2 weeks in Germany.
- How many modules are in the standard library for parsing HTML and working with text?

PHP was out because there are no functions for threads. C++ was out because I found it really hard to learn at that time and so I was afraid of needing much time to code. Perl was out because I really hated the syntax with its @-, dot- and other crazy symbols. At this time I knew Java quite good, so it was one language that I thought about to use. The other language was Python. I read an article in the Linux Magazine that introduced the language and said that the programmer’s productivity is very high.

Then I compared Java and Python – and finally choose Python. I did not recognize the advantages of Python compared to Java, I simply needed more time to develop in Java. Python was close to my thoughts – just after writing a few lines of code I saw hat I can express my ideas very easily with also less code than coding in Java.

TECHNICAL: LOADING OF WEBPAGES

The I started to use Python. With the Manual and my Editor (Kate from KDE project on SuSE Linux) I started implementing my idea. First I tried to use the asyncronous module called asyncore from the standard library. This worked fine up to 200 webpages – and then it became slower and slower.

So I tried to swith to threads – using the module “thread” and “threading” from the standard library. I had a really bad experience with threading in Python: I needed long to decide between “thread” and “threading”, for example. Please note: At this time I was quite new to threads and also to Python – I always learned programming languages by copying examples from a manual. Today I know what to do when I’ve got a problem, but at that time I simply had not enough background knowledge to understand some things in programming.

I also had another problem with threads: I tried to do each HTTP request in one thread, but they seemed to block each other, even if I only create five threads.

Then I switched to processes – they were my last chance. And wow: It got a system working in about three minutes [Example]. It worked fine, nothing blocking. And on my webserver I tried systems with 500 webpages loading each one a script that sleept for 3 seconds on another server. Everything was great: The program produced only a little overhead and took about 3,5 seconds to finish.

But the next problem came immediately: Forking so much processes and handling them can work fine on a fast machine, but if there are some parallel search queries or the machine is weak the mass of processes are getting a problem for performance.

So I tried a mixed up system: Using the “asyncore” module within the forked processes. This became the best solution and compromise between loading speed and usage of system resources: Today Metager2 splits the list of URLs of webpages that has been returned by the source search engines up into lists containing 50 webpages. Then for each URL package a new process is forked. This system seem to scale controlled up to thousands of webpages.

And processes have one big advantage: If there is an error in one because anything goes wrong, the main process can still use the contents returned by the other good-working processes. And: You can be sure that the loading process does not take to much time, simply by killing the child processes after a specified time from the parent process.

The reason why I don't show a benchmark here and why I always say "seem" is: The problem is the bandwidth. The performance of the loading system uses intensively the connection of your network provider which is much different at each hosting company. I don't want to do advertising here, but by moving the Metager2 webserver from Strato to Host Europe the *same* system ran 30 to 40% faster.

TECHNICAL: WHAT TO DO WITH THE CONTENTS

The next step was to write a system that ranks the retrieved contents. I looked for interesting modules in Python's library, and I found one. Because I formerly worked with PHP and Perl, I wanted to re-use my regular expressions. Metager2 does not use any markup parsers or whatever, it collects and analyses information with the module called "re". Later I splitted off a development version and tried started working with "htmllib" and other packages, but they were not as easy to use – and even not so fast. But this is also a personal preference of me – I find regular expressions simply more stylish.

The problem was: How to easily get the contents that were loaded in the forked processes and use them in the main process? I did not want to learn complex inter-process-communication. My solution was: The forked childs store the contents of each webpage that was loaded in temporary files. The main process loops until there are no more files. I know, this created much IO-access on the system's harddisk – but it is very stable, secure and simple to code.

Today Metager2 parallelizes the content extraction into several different threads in order to speed up this system.

So all in all: Loading is done with asynchronous sockets inside processes, and the analyse is done later within several threads.

TECHNICAL: SPAM FILTERS

In May 2005 then I published my baby Metager2 at a conference of the SuMa-eV in Hannover. In the next few months my work focused more and more on writing good spam filters that detect improper search results. Metager2 is the world's only search engine that always had the up-to-date contents of a website. The same contents that a user will see when one clicks on a search result.

So I invented content-based spam filters for Metager2, which became one main goal until today. The German computing magazine c't said in edition 10/2006 that in tests Metager2 delivered high-quality search results. There was almost no spam or improper results, they said.

An example of how stupid our source search engines like Google can be: Do a search for "HTTP" at Yahoo, Google or MSN. And see if you really get results for the one of the most important protocols.

Python helps me writing good spam filters because I can implement a new one very fast if necessary or to adjust existing one. I often get and email from someone using Metager2 that says to me: Why is this webpage filtered? Why this one not? Our spam filters are not perfect, but that’s no problem because with Python I can rapidly implement changes.

FUTURE WORK

In the next few months I will do license Metager2’s loading system and the spam filters to search engines. I am in talk with one big European search provider, and you will hear from us in a few months ☺

The other work I will be focused on is to write an own crawler. A metasearch engine always depends on source search engines. So if there is an own index to fall back if they stop allowing us to use their results is very important. So in the next months I will learn how to use C libraries or how to connect with Java because my eye is on Lucene / Nutch as the base system for an own index.

SUMMARY

…