Combinatorial Input Recognition Algorithm Internet Applications to HTTP Web Servers

Joseph K. Myers

Department of Mathematics and Statistics, College of Liberal Arts and Sciences

Abstract. We introduce an algorithm for the classical recognition problem of identifying elements s within a class of strings S = $\{s_1, s_2, ..., s_n\}$. Here it is assumed that struct s = { char *a; int z; }; and that for two strings s_1, s_2 if $s_1.z == s_2.z$ then there is J = $\{j_1, j_2, ..., j_l\}$ with l(z) < z so that $s_1[J] == s_2[J] \Rightarrow s_1 == s_2$. The goal is to minimize $k\sum_{i=1}^{n} l(s_i.z)$ and thus distinguish elements with the smallest number of comparisons. In general, this simply means that in a language system we face a nonoptimal set of keywords (n < sizeof(*s.a)z) and so we seek to reconstruct the language with a smaller set of k recognition vertices so that $k \approx \log_{\text{sizeof(*s.a)}} n$. The algorithm is applied to increasing HTTP transmission and request processing speed. The result is an increase in web server performance from 20,000 to 35,000 or more hits per second. The input recognition algorithm produces the greatest improvement when interpreting HTTP request headers which are large in proportion to the response. This is the case when collecting data for statistical analysis of Internet traffic.

1. Introduction

Our algorithm is a fast way of recognizing a set of keywords.

The most natural application of this algorithm is to simplify the data processing work, in particular, language token recognition, that is continuously going on behind the scenes as part of the text-based communications protocol that drives the Internet, namely HTTP. It is important that one notes the practical application of our algorithm is to a predetermined—a standardized—set of keywords and commands. It is our belief that dynamical recognition systems—binary trees, sorting structures, hash tables—are misapplied and waste Internet resources when used to translate the standardized form of requests defined in RFC 2616. Furthermore, using a static system enables one to identify input errors exactly and without an increase in processing time, so it achieves absolute safety.

As server loads on the Internet becomes more and more intense, they are also becoming more and more optimized. Through universally beneficial methods like HTTP pipelining, less work is being done by both the client's web browser and by the HTTP web server.

Also, less operations are being performed by slow mechanisms such as sscanf(), and server administrators are searching for ways of processing thousands of requests in less than one second.

Thus the proportion of time spent to identify the prefix of each line within the HTTP request header, as well as the beginning and end of the header string, is more and more a critical measure of performance. This job of the HTTP server is performed first with byte-wise scanning to identify newline characters, then by lexical scanning to find known tokens at the beginning of lines.

The length of the HTTP pipeline can grow indefinitely, and as it does so, performance keeps increasing until it reaches the highest possible performance of the operating system. For instance, the fastest way to load a common web page with 98 images and scripts, is to concatenate the HTTP requests and send them as a single packet through the Internet to the HTTP server. System calls, Internet transmission costs, and browser rendering time are all saved as a result.

Yet at the same time, the lexical scanning function becomes of paramount importance. Out-of-date behavior, such as CRLF sequences and folded header lines, produces horrible nightmares within a system that has otherwise been able to develop efficiently.

No longer is it possible to implement them prudently when one must expect the single initial request to be followed directly by requests for all of the web page's prerequisites.

Most of all, the 98 subsequent requests are all the same. The item requested is the only varying parameter, and yet the typical browser sends a conflagration of repetitious data values; yes, even when it has repeated them 99 times.

So much redundancy practically begs for optimization which has been analyzed carefully. Silently, the situation is overwhelming the Internet. There is a need for an algorithm which finds the best value of k.

2. Discussion and results.

The algorithm is derived from an integer-labeling mechanism which reduces the number of textual comparisons to a value somewhat larger than the optimal value of k. The inspiration for this technique came from the idea that, if written in base N, any set of N elements could be identified by only one digit. Five servers were tested, the last of which uses the input recognition algorithm for scanning and interpreting the HTTP headers of clients' requests. The HTTP requests were identical to those produced by Microsoft Internet Explorer 6.0, the predominant browser in use today.

The graph shows the servers' hits/second as pipeline length is increased from 1 to 30. The results are from the latest versions of Apache, the most recent public release of Mathopd, and from "httpserver-0.1" which implements the recognition algorithm.

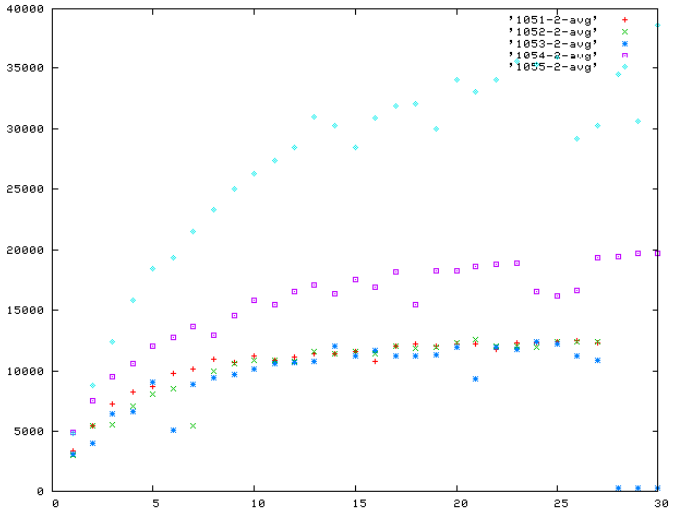

Figure 1. Hits per second vs. pipeline length on FreeBSD 6.1.

Legend: 1051 = Apache 1.3.37, 1052 = Apache 2.0.59, 1053 = Apache 2.2.4, 1054 = Mathopd 1.5p5, 1055 = httpserver-0.1 (with recognition algorithm).

3. Conclusion

We have found evidence that increasing pipeline length increases the performance of webservers. However, the theoretical limit of performance is diminished significantly by slow input recognition algorithms.

Up until the present, this area of performance may have been ignored due to the fact that most webservers have insufficient network connectivity for such improvements to have a practical effect. Nevertheless, network connectivity continues to reach greater levels of performance, and soon the input recognition speed will become a more significant limiting factor of webserver performance. Indeed, there is a greatly increasing demand for multiple server programs running simultaneously on a single webserver computer. In this situation, the performance is completely a matter of the server programs' data processing speed, and not the speed of the server computer's Internet connection.

By using the input recognition algorithm, we found an improvement in the processing speed of large request headers, such as those used in collecting statistics and performing analysis on Internet traffic.

4. Acknowledgments

The author would like to thank Dr. Dharam Chopra for his willingness to accept this topic for a project in his combinatorics class in fall 2006.