IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4692

Interactive URL Correction for Proxy Request

Kai-Hsiang Yang, Chi-Chien Pan, and Tzao-Lin Lee
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.
{ f6526004, d5526001, tl_lee}@csie.ntu.edu.tw

ABSTRACT

Proxy servers are getting more and more important today. They provide web page caches for users to browse quickly, and also reduce unnecessary network traffic. However, users can't browse web pages with the wrong URL. Sometimes we just want to get some information about some subjects, the proxy server couldn't help us at all. Most people often use search engines to find data, but they still have to type the correct URLs of search engines. We actually need the proxy server with the ability of "Interactive URL Correction", which means the proxy server could correct the URL request, and take the users to where they want to browse, or send back some possible URLs. Users simply enter one word "google", or even "goggle", and then will eventually be taken to "www.google.com". To accomplish the URL correction, we have applied URL preprocess and approximate URL matching technique into proxy server. In this paper, we implement the system on the "squid" proxy system , and use "edit distance" as the URL error measurement. Additionally, we also list the limitation of proxy parameters and the benefits of our system.

INTRODUCTION

With the rapid expansion of the World Wide Web (WWW) too many web-based applications had caused serious performance degradation on the Internet. Caching is the process of storing web elements (pages, files, images) on proxy servers. The use of caching has proliferated because it reduces bottlenecks. The Internet Caching Resource Center (www.caching.com) estimates that caching can reduce the need for bandwidth by at least 35 percent. Consequently, the proxy servers had been widely deployed to reduce the bandwidth for the same "web page" requests; proxy server could accelerate the browsing rate by storing current web pages for the future requests. When some web pages are very popular, the proxy server only needs to download them once, then users could quickly browse these pages from the proxy. Nowadays, proxy server becomes necessary for the WWW community.

When proxy server receives a web page request (called "URL request"), it first matches all web pages in its native database. If the URL request is correct (by the DNS lookup), the proxy server immediately sends the requested page back to the user. Otherwise it has to access the requested page through the Internet, and then sends the page back and stores it in its native database. However, this is inconvenient and insufficient for users. When users browse web pages, they sometimes enter the wrong URL or just guess one URL for the product or company name. For example, we often enter the URL ("www.starbucks.com") for the company "Starbucks". But when we enter the word with some errors, for example "www.starbuck.com", the proxy will return error messages to us, and doesn't help us any more. Users have to correct the URL by themselves.

Most people often use search engines to find data, but they still have to enter correct URLs of search engines. We actually need to have the proxy server with the ability of "Interactive URL Correction", which means the proxy server could correct the URL request, and take the users to where they want to browse, or send back some possible URLs for users to choose the correct URL by simply one click. Users could just enter one word "google", or error word "goggle", and then will be taken to "www.google.com".

To accomplish the URL correction, we have applied URL preprocess and approximate URL matching technique into proxy server, so that users could just enter some important names (maybe with some errors) to browse the web pages they want. Normally, the proxy server works as usual when the URL request is correct. However if proxy server discovers the URL request is non-existent, it first performs the URL preprocess to get the important part of URL, and then performs the approximate URL matching to obtain some approximate URLs, and sends back to the users. Users could see the approximate URLs listed in their browsers and just simply click the correct URL to browse the web page. This is very convenient for users to browse on the Internet.

In this paper, we choose "edit distance" as the URL similarity measurement of two URLs, because it has a clear definition and is also widely used in many fields of applications. Furthermore, we have designed one algorithm to utilize three filter conditions [14] based on n-gram technique to perform the URL correction.

This paper is organized as follows: Section 2 presents related work, Section 3 lists some basic concepts about our method, Section 4 outlines the design of the URL correction, Section 5 presents the implementation environment and results, and the last section is the conclusion.

RELATED WORK

Web tracking and caching is highly active research area. A lot of tracking studies analyze the request rate, number of requests, the effects of cookies, aborted connections, and persistent connections on the performance of proxy caching [2, 3].

There has also been extensive work on cooperative Web caching as a technique to reduce access latency and bandwidth consumption. Cooperative Web caching proposals include hierarchical schemes like Harvest and Squid [4, 5], hash-based schemes [6], directory-based schemes [7] and multicast-based schemes [8].

For the approximate matching field, many researches have been published. For two strings of length n and m, there exists a dynamic programming algorithm to compute the edit distance of the strings in O(nm) time and space [9], and improvements to the average and worst case have appeared [10, 11].

In [1], they solve the problem of approximate string joins in a database, using n-gram as index stored in database and using three filter conditions for quickly joins. In the field of database, several indexing techniques proposed for the "approximate string matching" problem, however, such techniques have to be supported by the database management system [12].

BASIC CONCEPTS

In this section, we briefly describe some basic concepts about URL preprocess and URL approximate matching technique.

Edit Distance (The URL Similarity Measurement)

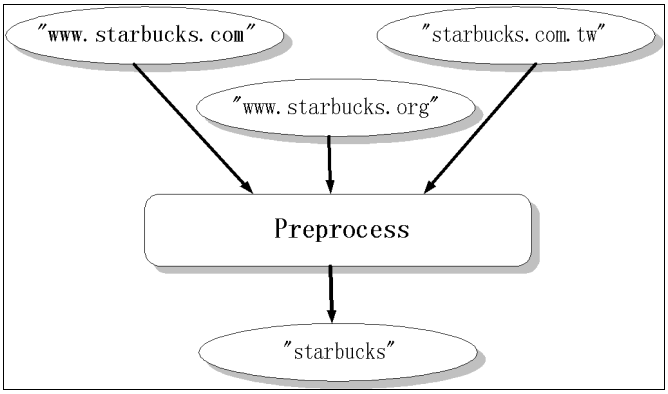
The edit distance d(x,y) between two URLs x and y is the minimal cost of a sequence of operations that transform x into y. The cost of a sequence of operations is the sum of the costs of the individual operations. In this paper we use three standard operations of cost 1 such as follows.

Insertion: inserting the letter a, Deletion: deleting the letter a, Replacement or Substitution: for a≠b, replacing a by b.

URL Preprocess

Before we have to introduce the index (called n-grams) for each URL, we perform one preprocess. The preprocess procedure prunes some common prefixes or suffixes of each URL, such as: "www.", ".com", ".org", ".gov", ".tw", etc. For all URLs in proxy native database we must perform this prepro-

Figure 1. The preprocess prunes URL strings into some important keyword



cess first, and then start to make the n-grams of the pruned URLs. On the other hand, we also perform the same preprocess for all URL request before matching. The benefit of the procedure is that users could just type some important parts of the URL, and they don't need to consider what the prefix or suffix is. The following example shows the influence.

Example [Preprocess] As Figure 1 shows, assume proxy server has three URL strings about "starbucks" (such as: "www.starbucks.com", "www.starbucks.org", "starbucks.com.tw",) then the preprocess prunes the three URLs to the same string "starbucks". Therefore users could just enter "starbucks" to find these three URLs. The improvement is very convenient for users.

N-Grams: Indices for Approximate URL Matching

For a given pruned URL s, its n-grams are obtained by "sliding" a window of length n over the characters of s. Since n-grams at the beginning and the end of the string have fewer than n characters from s, we introduce new characters "#" and "$", and conceptually extend the string by prefixing it with occurrences of "#" and suffixing it with occurrences of "$". Thus, each n-gram contains exactly n characters. The concept behind using n-grams is that when two strings a, b are within a small edit distance of each other, they must share a large number of n-grams in common.

For any string s of length |s|, we can easily find out the number of its n-gram is |s| + n –1. For example, the pruned URL s is "DIGITAL", and then its n-grams are: "##D", "#DI", "DIG", "IGI", "GIT", "ITA", "TAL", "AL$", "L$$". The number of 3-grams: 9 = 7 (length) + 3(n) – 1.

Filtering Technique Using N-Gram

For a large URL cache in proxy, we use three filter conditions [1] to quickly filter out impossible URLs having edit distance less then k (k is the error threshold of proxy). The key objective here is to efficiently identify approximate URLs before we use the "expansive" distance function to compute their distance. These three filtering conditions are as follows:

Count Filtering: Consider strings s_1 and s_2, of lengths $|s_1|$ and $|s_2|$, respectively. If the equation $d(s_1, s_2) \leq k$ holds, then the two strings must have at least $(\max(|s1|) - 1 - (k - 1)*n)$ the same n-grams.

Position Filtering: If strings s_1 and s_2 are within an edit distance of k, then a positional n-gram in one cannot correspond to a positional n-gram in the other that differs from it by more than k positions.

Length Filtering: The last condition is that string length provides useful information to quickly prune strings that are not within the desired edit distance. If two strings s_1 and s_2 are within edit distance k, their lengths cannot differ by more then k.

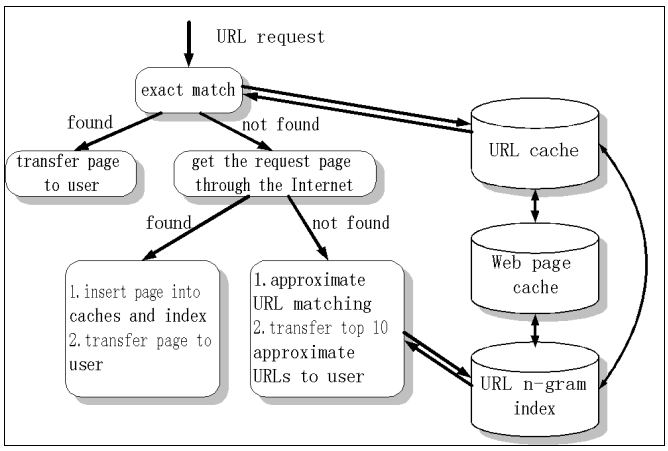
URL CORRECTION DESIGN

In this section we introduce our method and new proxy architecture for the URL correction.

New Proxy Architecture

We modify the proxy architecture to perform approximate URL matching, and Figure 2 shows the new architecture of proxy server. We need to make

Figure 3. Matching process



a new "URL N-gram Index" in addition to the "web page cache" and corresponding "URL cache"; the "URL N-gram Index" is the set of all n-grams of each URL in proxy. Especially, we apply the approximate URL matching into the situation when proxy server couldn't get the requested page, then the proxy server returns top 10 approximate URLs back to the user.

Index Architecture

For each web page, proxy server stores it into inside web page cache and URL cache. Furthermore, we create the n-grams for each URL and use the set of n-gram (G_s) as the URL indices. We put the indices G_s into a large table (called "URL N-gram Index"). The URL N-gram Index contains four fields: 1.n-gram 2.URL string length (denote L) 3.position (the position which n-gram appears) 4.URL_ID (the unique identification of each URL).

Example [URL N-gram Index] Assume that URL string D_s = "HELLO", Length(D_s)= |Ds| = 5, and we use the 3-grams as indices(n = 3,) then we get the following 3-grams:

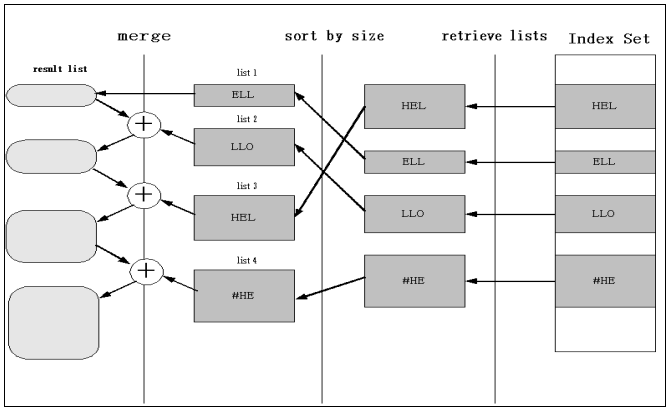
$G_{3,1}$ = *"##H"*, $G_{3,2}$ = *"HE"*, $G_{3,3}$ = *"HEL"*, $G_{3,4}$ = *"ELL"*, $G_{3,5}$ = *"LLO"*, $G_{3,6}$ = *"LO$"*, $G_{3,7}$ = *"O$$"*

Matching Processes

The URL correction processes using n-gram are as follows:

1. For each URL string D_s, we produce all the n-grams of D_s.
2. Retrieve each filter list in the URL N-gram Index corresponding to each n-gram.
3. In all filter lists, we sum the records having the same URL_ID. When the sum is greater then the **Count Filtering**, the record with the URL_ID maybe is the answer; then we check it for the **Length Filtering**, and insert it into the last result list when it passes the condition.
4. Use the distance function to compute the edit distance for the records in the last result list.

Figure 2. The new proxy architecture



Some serious problems arise during these processes, especially when the amount of record in filter lists is very large. Therefore we need an efficient method for these merge processes. We sort records in each filter list by URL_ID field, like the merge-sort algorithm. The following j iterations present the method:

1. List 1 => Result List (initiation).
2. List 2 + Previous Result List => new Result List (because we sort the records by URL_ID in lists, we can do the counting linearly in time O(n)).
3. List j + Previous Result List => new Result List.

During the merge iterations, we can easily observe that the preceding list records also appear in the latter lists, and the space and time used for counting increases quite substantially. For the purpose to reduce the space and time, we sort all lists by size beforehand, and the first list has the smallest size. Figure 3 shows our searching processes and data structures.

Rank and Return

After these matching processes, we have the real distances of last few possible URLs. Then we rank the approximate URLs by its similarity with the requested URL; we report error messages and the top 10 URLs back to users.

IMPLEMENTATION AND RESULTS

In this section, we present the implementation environment, browser setting, and parameter limitation of proxy server. Furthermore, we also describe the performance and benefits of our implementation.

Environment

We implement our proxy server on the Linux platform (Red Hat Linux release 6.2,) and choose the "squid" proxy system to apply URL correction technique, because the system is an open source and widely used under most network infrastructures. Besides, we also use the DB library developed by the University of Berkeley to perform the b+ tree structures for the URL N-gram Index.

Our implementations contain two parts:

(1) *Index Generation*

The part is responsible for generating n-grams of all URLs and making the sorted URL N-gram Index mentioned above, and sorting lists, etc. We use 3-gram (n=3) as default in our implementation.

(2) *Filter and Matching*

Programs could match approximate URLs under k edit distance, and we set k = 2 in the proxy settings.

For the distance function, we use the Levenshtein distance algorithm [13] to compute the real distance between two URL strings.

Browser Setting

In order to have faster efficiency on browsing the web, it is necessary for the user to set the proxy server in the web browser. This helps to make more efficient use of bandwidth and reduce the chances of getting duplicated copies of the same data from overseas. Two common browsers, Netscape Navigator and Internet Explorer, have to be configured to use the proxy server; especially in the IE browser, we have to check the check box of "*Access the Internet using a proxy server*" and cancel "*Bypass proxy server for local (Intranet) addresses*". The later action is very important, because the IE browser would automatically append local domain to the requested URL when it is just one word; if we don't cancel the later check box, the proxy server would not receive any URL request.

Parameter Limitation

In our implementation, the proxy server could select different parameters (n-gram, k error threshold) to work, however, the filters would lose their functionality when we choose unsuitable parameters. The limitation comes from the **Count Filtering:** $L - 1 - (k - 1)* n > 0$.. That is, our filters will lose functions when the inequality doesn't hold. In experiments, we choose k = 2 and n = 3 for proxy server, therefore, the filters work well for the pruned URL strings of length L > 4. For strings of length L <= 4, we have to directly compute their edit distances.

Experimental Results

We used about 500,000 URL strings to evaluate the performance of approximate URL matching, and produce more then 5,000,000 n-gram data. In our experiments, almost approximate URL matching processes had finished in one to three seconds; the performance of the filtering is acceptable. As our previous research [14], we use the matching processes to perform the filtering. However, the n value is very important for search performance. If n is too large, the filters lose its functions, then we have to use brute force method to check each string, and the performance decreases. If n is too small, the index size increases, and the performance also decreases. Therefore, it is very important to one suitable n value, and in our experience, n (three to five) is suitable for common situations.

We could change the proxy parameters for various approximate levels depending on different needs. On the other hand, users could just enter some important keywords to find what they want because we first perform the URL preprocess, and maybe users would discover some other URLs containing the information they are interested in.

CONCLUSIONS

We successfully applied the URL correction technique into proxy server, and this kind of proxy server will take users to a convenient environment for browsing on the Internet. Even though users enter error URLs, they still will be taken to the correct web pages. This is our major contribution.

To perform the URL correction technique, we make and sort the n-grams of all URL strings, and archive a well URL correction performance. Furthermore, we list the limitation of the proxy parameters for administrators to customize the system. Most of all, we make users a lot easier to browse the web.

To increase the practicality of the system, it should be deployed on one bigger proxy server, such as some ISP's proxy servers, which has a lot of web pages and a lot of user requests.

REFERENCES

[1] L. Gravano and P. G. Ipeirotis and H. V. Jagadish and N. Koudas and S. Muthukrishnan and D. Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th VLDB Conference, 2001*.

[2] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: The devil is in the details. In Workshop on Internet Server Performance, pages 111-118, June 1998.

[3] A. Feldmann, R. Cacres, F. Douglis, G. Glass, and M. Rabinovich. Performance of web proxy caching in heterogeneous bandwidth environments. In *Proceedings of IEEE INFOCOM '99, March 1999*.

[4] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical Internet object cache. In *Proceedings of the 1996 USENIX Technical Conference, pages 153-163, January 1996*.

[5] Squid internet object cache. http://squid.nlanr.net.

[6] D. Karger, T. Leighton, D. Lewin, and A. Sherman. Web caching with consistent hashing. In *Proceedings of the 8th Int. World Wide Web Conference, May 1999*.

[7] R. Tewari, M. Dahlin, H. Vin, and J. Kay. Design considerations for distributed caching on the Internet. In the 19th IEEE Int. Conference on Distributed Computing Systems, May 1999.

[8] J. Touch. The LSAM proxy cache – a multicast distributed virtual cache. In *Proceedings of the 3rd Int. WWW Caching Workshop, June 1998*.

[9] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, 147: pages 195-197, 1981.

[10] R. Cole and R. Hariharan. Approximate string matching: a simpler faster algorithm. In *Proceedings of ACM-SIAM SODA'98*, pages 463-472, 1998.

[11] W. Chang and E. Lawler. Sublinear approximate string matching and biological applications. Algorithmica, 12(4/5):327-344, 1994. Preliminary version in FOCS'90, 1990.

[12] T. Bozkaya and Z. M. Ozsoyoglu. Distance based indexing for high dimensional metric spaces. In *Proceedings of String Processing and Information Retrieval Symposium (SPIRE'99,)* pages 16-23, 1999.

[13] Levenshtein Distance. http://www.merriampark.com/ld.htm

[14] Chi-Chien Pan and Kai-Hsiang Yang and Tzao-Lin Lee. Approximate String Matching in LDAP based on edit distance. In *Proceedings of the IPDPS2002 Conference, 2002*.