

Techniques for Efficient Fragment Detection in Web Pages

Lakshmith Ramaswamy^{1*}

Arun Iyengar²

Ling Liu¹

Fred Douglass²

¹ College of Computing, Georgia Tech
801 Atlantic Drive
Atlanta GA 30332
{laks, lingliu}@cc.gatech.edu

² IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{aruni, fdouglass}@us.ibm.com

ABSTRACT

The existing approaches to fragment-based publishing, delivery and caching of web pages assume that the web pages are manually fragmented at their respective web sites. However manual fragmentation of web pages is expensive, error prone, and not scalable. This paper proposes a novel scheme to automatically detect and flag possible fragments in a web site. Our approach is based on an analysis of the web pages dynamically generated at given web sites with respect to their information sharing behavior, personalization characteristics and change patterns.

Keywords

Fragment-based publishing, Fragment caching, Fragment detection

1. INTRODUCTION

Dynamic content on the web has posed serious challenges to the scalability of the web in general and the performance of individual web sites in particular. There has been considerable research towards alleviating this problem. One promising research direction that has been pursued and successfully commercialized in recent years is *Fragment* based publishing, delivery and caching of web pages [2, 7, 8].

Research on fragment-based publishing and caching has been prompted by the following observations:

- Web pages don't always have a single theme or functionality. Often web pages have several pieces of information, whose themes and functions are independent.
- Generally, web pages aren't completely dynamic or personalized. Often the dynamic and personalized content are embedded in relatively static web pages.
- Web pages from the same web site tend to share information among themselves.

*Most of this work was done while Lakshmith was an intern at IBM Research in the summers of 2002 and 2003

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CIKM 2003 November 3–8, 2003, New Orleans, Louisiana, USA.
Copyright 2003 ACM 1-58113-723-0/03/0011 ...\$5.00.

Conceptually a fragment can be defined as a part of a web page (or more generally part of another fragment), which has a distinct theme or functionality associated with it and which is distinguishable from the other parts of the web page. In fragment-based publishing, the cacheability and the lifetime are specified at a fragment level rather than the page level.

The advantages of fragment-based schemes are apparent and have been conclusively demonstrated in the literature [7, 8]. By separating the non-personalized content from the personalized content and marking them as such, it increases the cacheable content of the web sites. Furthermore, with fragment-based publishing, the amount of data that gets invalidated at the caches is reduced. In addition, the information that is shared across web pages needs to be stored only once, which improves the disk space utilization at the caches.

Though there have been considerable research efforts on performance and benefits of fragment-based publishing and caching, there has been little research on detecting such fragments in existing web sites. Most of the research efforts on fragment caching rely upon the web administrator or the web page designer to manually fragment the pages on the web site. However, manual fragment detection is both very costly and error prone.

In this paper, we propose a novel scheme to automatically detect and flag possible fragments in a web site. We analyze web pages with respect to their information sharing behavior, personalization characteristics, and the changes occurring to them over time. Based on this analysis, our system detects and flags the “interesting” fragments in a web site. The research contributions of this paper are along three dimensions. First, we formally define the concept of a *Candidate Fragment*, which forms the basis for fragment detection schemes. Second, we provide an infrastructure for detecting fragments within a web site, including an efficient document model and fast string encoding algorithm for fragment detection. Third, we present two algorithms for detecting fragments that are shared among M documents or that have different lifetime characteristics, which we call the *Shared Fragment Detection Algorithm* and *Lifetime-Personalization based (L-P) Fragment Detection Algorithm* respectively.

2. A FRAGMENT MODEL OF WEB PAGES

The web documents considered in this paper are HTML documents. We assume that all HTML documents are *well-*

formed [6]. Documents that are not well formed can be converted to well-formed documents. We refer to such a transformation as document normalization. HTML Tidy [3] is a well-known Internet tool for transforming an arbitrary HTML document into a well-formed one.

2.1 Candidate Fragments

We introduce the notion of candidate fragments as follows:

- Each Web page of a web site is a candidate fragment.
- A part of a candidate fragment is itself a candidate fragment if any one of the following two conditions are satisfied:
 - The part is shared among “M” already existing candidate fragments, where $M > 1$
 - The part has different personalization and lifetime characteristics than those of its encompassing (parent or ancestor) candidate fragment.

It is evident from the definition that the two conditions are independent and these conditions define fragments that benefit caching from two different and independent perspectives. We call the fragments satisfying Condition 1 **Shared fragments**, and the fragments satisfying Condition 2 **L-P fragments** (denoting Lifetime-Personalization based fragments). Lifetime characteristics of a fragment govern the time duration for which the fragment, if cached, would stay fresh (in tune with the value at the server). The personalization characteristics of a fragment correspond to the variations of the fragment in relation to cookies or parameters of the URL.

It can be observed that the two independent conditions in the candidate fragment definition correspond well to two key aims of fragment caching. By identifying and creating fragments out of the parts that are shared across more than one fragment, we avoid unnecessary duplication of information at the caches. By creating fragments that have different lifetime and personalization properties we not only improve the cacheable content but also minimize the amount and frequency of the information that needs to be invalidated.

3. FRAGMENT DETECTION: THE BASICS

The primary goal of our system is to detect and flag candidate fragments from pages of a given web site. The fragment detection process is divided into three steps. First, the system is conceived to construct an *Augmented Fragment Tree* (see Section 3.1) for the pages of a web site. Second, the system applies the fragment detection algorithms to augmented fragment trees of the web pages to detect the candidate fragments. In the third step, the system collects statistics about the fragments such as the size, how many pages share the fragment, access rates etc. These statistics aid the administrator to decide whether to turn on the fragmentation. Figure 1 gives a sketch of the architecture of our fragment detection system.

Our fragment detection framework has two independent schemes: a scheme to detect Shared fragments and another scheme to detect L-P fragments. Both of the schemes are located in one large framework, probably collocated with a server-side cache, and work on the web page dumps from the web site.

The scheme to detect Shared fragments works on various pages from the same web site, whereas the L-P fragments

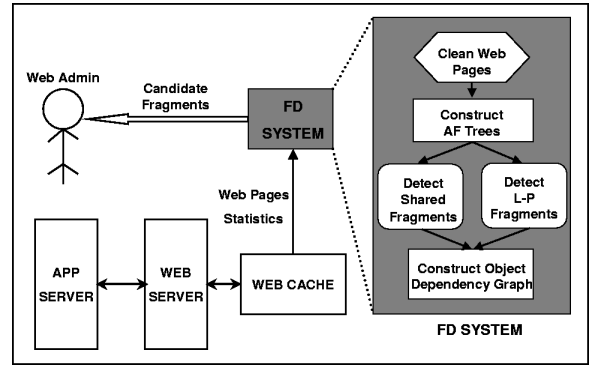


Figure 1: Fragment Detection System Architecture

approach works on different versions of each web page (if the web page has more than one version). For example, in order to detect L-P fragments, we have to locate parts of a fragment that have different lifetime and personalization characteristics. This can be done by comparing different versions of the web page and detecting the parts that have changed and the parts that have remained constant. However, to detect Shared fragments, we are looking for parts of a fragment that are shared by other fragments, which naturally leads us to work with a collection of different web pages from the same web site.

While the inputs to the L-P fragment detection scheme and the Shared fragment detection approach differ, both schemes rely upon the Augmented Fragment Tree representation of its input web pages, which is described in the next subsection. The output of our fragment detection system is a set of fragments that are shared among a given number of documents or that have different lifetime characteristics. This information will be served as recommendations to the fragment caching policy manager or the respective web administrator (see Figure 1).

3.1 Augmented Fragment Tree

A good model to represent the web pages is one of the keys to efficient and accurate fragment detection. We introduce the concept of an Augmented Fragment Tree as a model to represent web pages.

An augmented fragment (AF) tree is a hierarchical representation of the structure of an HTML document. It is a compact DOM tree [1] with all the text-formatting tags removed, and each node augmented with additional information for efficient comparison of different documents and different fragments of documents. Each node in the tree is annotated with the following fields:

- Node Identifier (Node-ID): A vector indicating the location of the node in the tree.
- Node-Value: A string indicating the value of the node. The value of a leaf node is the text itself and the value of an internal node is NULL (empty string).
- Subtree-Value: A string that is defined recursively. For a leaf node, the Subtree-Value is equal to its Node-Value. For all internal nodes, the Subtree-Value is a concatenation of the Subtree-Values of all its children nodes and its own Node-Value. The Subtree-Value of a node can be perceived as the fragment (content region)

of a web document anchored at this subtree node.

- **Subtree-Size:** An integer whose value is the length of Subtree-Value in bytes. This represents the size of the structure in the document being represented by this node.
- **Subtree-Signature:** An encoding of the subtree value for fast comparison. We choose shingles [5, 9] as the encoding mechanism (see the discussion below). Therefore we also refer to the Subtree-Signature as Subtree-Shingle.

Shingles [5, 9] are essentially fingerprints of the document (or equivalently a string). But unlike other fingerprints like MD5, if the document changes by a small amount, its shingle also changes by a small amount.

Figure 2 illustrates this property by giving examples of the MD5 hash and the shingles of two strings. The first and the second strings in the figure are essentially the same strings with small perturbations. It can be seen that the MD5 hash of the two strings are totally different, whereas the shingles of the two strings vary just by a single value out of the 8 values in the shingles set (shingle values appearing in both sets are underlined in the diagram). This property of shingles has made it very popular in estimating the resemblance and containment of documents [5, 4].

Fragment based publishing of web pages improves the scalability of web services. In this paper we provide efficient techniques to automatically detect fragments in web pages. We believe that automating fragment detection is crucial for the success of fragment based web page publication.

MD5: 982f3bb69a174efb0aa4135c99e30d04

Shingles: {801384, 896252, 1104260, 1329558, 1476690, 1569872, 1772039, 2001370}

Fragment based publishing of web pages improves the efficiency of web services. In this paper we provide scalable techniques for automatic detection of fragments in web pages. We believe that automating fragment detection is critical for the success of fragment based web page publication.

MD5: 91d16c3e9aee060c82c626d7062d0165

Shingles: {801384, 896252, 1104260, 1476690, 1569872, 1772039, 2001370, 2033430}

Figure 2: Example of Shingle Vs MD5

3.2 AF Tree Construction

The first step of our fragment detection process is to convert web pages to their corresponding AF trees. The AF tree can be constructed in two steps. The first step is to transform a web document to its DOM tree and prune the fragment tree by eliminating the text formatting nodes. The result of the first step is a specialized DOM tree that contains only the content structure tags (e.g., like <TABLE>, <TR>, <P>). The second step is to annotate the fragment tree obtained in the first step with Node-ID, Node-Value, Subtree-Value, and Subtree-Shingle.

4. FRAGMENT DETECTION ALGORITHMS

Having described the structure of the AF tree and methodology to construct it, we now describe the algorithms to detect Shared and L-P fragments.

4.1 Detecting Shared Fragments

The Shared fragment detection algorithm operates on various web pages from the same web site and detects candidate fragments that are “approximately” shared. In our framework for Shared fragment detection, we add three additional parameters to define the appropriateness of such approximately shared fragments. These parameters can be configured based on the needs of the particular web site. The accuracy and the performance of the algorithm are dependent on the values of these parameters.

- **Minimum Fragment Size** (*MinFragSize*): This parameter specifies the minimum size of the detected fragment.
- **Sharing Factor** (*ShareFactor*): This indicates the minimum number of pages that should share a segment in order for it to be declared a fragment.
- **Minimum Matching Factor** (*MinMatchFactor*): This parameter specifies the minimum overlap between the Subtree-Shingles to be considered as a shared fragment.

The shared fragment detection algorithm performs the detection in three steps. First, it creates a pool of all the nodes belonging to AF trees of every web page of the web site, removing all nodes that do not meet the *MinFragSize* threshold. Then the algorithm processes each node in the pool in decreasing order of their sizes. While processing each node, it is compared against other nodes in the pool and groups the nodes that are *similar*. The similarity between nodes is measured by comparing their *SubtreeShingles*. If the number of nodes a group has is equal or higher than *ShareFactor*, then that group of nodes is flagged as a candidate fragment. The third step ensures that the scheme detects only the fragments that are maximally shared by eliminating those fragments that are not maximally shared. This is done by checking whether there was a larger fragment which has already been detected that contained the ancestors of all the nodes in the current group and no other nodes. If such a fragment has already been detected, then this is not a maximally shared fragment and hence not declared as a candidate fragment. Otherwise it is declared as a fragment, and all the nodes in the group are removed from the nodes pool.

4.2 Detecting L-P Fragments

We detect the L-P fragments by comparing different versions of already existing candidate fragments (web pages) and identifying the changes occurring among the different versions.

The input to this algorithm is a set of AF trees corresponding to different versions of web pages. These versions may be time-spaced or versions generated with different cookies.

The nodes of the AF trees in this algorithm have an additional field termed as the *NodeStatus*, which can take any value from {*UnChanged*, *ValueChanged*, *PositionChanged*}.

The scheme compares two versions of a web page at each step and detects L-P candidate fragments. Each step outputs a set of candidate fragments, which are merged to obtain the Object Dependency Graph (ODG) of the entire document. Object Dependency Graph [7] is a graphical

representation of the containment relationship between the fragments of a web site. Each step of the algorithm executes in two passes. In the first pass the algorithm marks the nodes that have changed in value or in position between the two versions of the AF tree. This is done by recursively comparing each node of the AF tree of one version with the most similar node from the AF tree of the second version. The content and the position of the two nodes are compared and accordingly the *NodeStatus* of the nodes are marked as *ValueChanged*, *PositionChanged* or *Unchanged*. The second pass of the algorithm detects the candidate fragments and merges them to obtain an Object Dependency Graph.

Similar to the Shared fragment detection algorithm, we have a few configurable parameters in this algorithm:

- **Minimum Fragment Size**(*MinFragSize*): This parameter indicates the minimum size of the detected fragment.
- **Child Change Threshold**(*ChildChangeThreshold*): This parameter indicates the minimum fraction of children of a node that should change in value before the parent node itself can be declared as *ValueChanged*.

5. EXPERIMENTAL EVALUATION

We have performed a range of experiments to evaluate our automatic fragment detection scheme. In this section we give a brief overview of two sets of experiments. The first set of experiments tests the two fragment detection algorithms, showing the benefits and effectiveness of the algorithms. The second set studies the impact of the fragments detected by our system on improving the caching efficiency.

The input to the schemes is a collection of web pages including different versions of each page. Therefore we periodically fetched web pages from different web sites like BBC (<http://news.bbc.co.uk>), Internetnews (<http://www.internetnews.com>), Slashdot (<http://www.slashdot.org>) etc. and created a web ‘dump’ for each web site.

In the first set of experiments we evaluated our shared fragment detection scheme and the impact of the parameters *MinMatchFactor* and *MinFragSize* on the detected shared fragments. For example on a dataset of 75 web pages from BBC website, which were collected on the 14th of July 2002, our shared fragment detection algorithm detected 350 fragments when *MinFragSize* was set to 30 bytes and *MinMatchFactor* was set to 70%. In all our experiments we noticed that our algorithm detected a larger number of smaller sized fragments when *MinMatchFactor* was set to higher values. For the same BBC dataset, the number of fragments increased to 358 when the *MinMatchFactor* was increased to 90%. In our experiments we also noticed that a large percentage of detected fragments are shared by 2 pages and only a few fragments are shared by more than 50% of the web pages.

Our second experiment was aimed at studying the performance of the L-P fragment detection algorithm. Though we experimented with a number of web sites, due to space limitations, we briefly discuss our experiments on the web site from Slashdot (<http://www.slashdot.org>). A total of 79 fragments were detected when the *ChildChangeThreshold* was set to 0.50, and 285 fragments were detected when *ChildChangeThreshold* was set to 0.70. We observed higher numbers of fragments being detected when

ChildChangeThreshold is set to higher values in all our experiments.

In our final set of experiments we study the impact of fragment caching on the performance of the cache, the server and the network when the web sites incorporate the fragments detected by our system into their respective web pages.

Incorporating the detected fragments improves the performance of the caches and the web servers in at least two ways. First, as the information that is shared among web pages is stored only once, the disk-space required to store the data from the web site is reduced. For example, our experiments on the BBC web site show that disk space requirements are reduced by around 22% by using the fragments detected by our algorithm.

Secondly, incorporating fragments into web sites reduces the amount of data invalidated in the caches. This in turn causes a reduction in the traffic between the origin servers and the cache. Our experiments show that for each web site this reduction is closely related to the average number of fragments in the web pages, their invalidation rates and the request rates to the web pages in the web site. However, the amount of data transferred between server and cache in a page-level caching scheme is higher than the data transferred between server and cache in a fragment-level caching scheme.

6. CONCLUSION

This paper addresses the problem of automatic fragment detection in web pages. We provided a formal definition of a fragment and proposed an infrastructure to detect fragments in web sites. Two algorithms are developed for automatic detection of fragments that are shared across web pages and fragments that have distinct lifetime and personalization characteristics. We report the evaluation of the proposed scheme through a series of experiments, showing the effectiveness of the proposed algorithms.

7. REFERENCES

- [1] Document object model - w3c recommendation. <http://www.w3.org/DOM>.
- [2] Edge Side Includes Standard Specification. <http://www.esi.org>.
- [3] Html tidy. <http://www.w3.org/People/Raggett/tidy/>.
- [4] Z. Bar-Yossef and S. Rajagopalan. Template Detection via Data Mining and its Applications. In *Proceedings of WWW-2002*, May 2002.
- [5] A. Broder. On resemblance and Containment of Documents. In *Proceedings of SEQUENCES-97*, 1997.
- [6] D. Buttler and L. Liu. A Fully Automated Object Extraction System for the World Wide Web. In *Proceedings of ICDCS-2001*, 2001.
- [7] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, May 2000.
- [8] A. Datta, K. Dutta, H. Thomas, D. VanderMeer, Suresha, and K. Ramamritham. Proxy-Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation. In *Proceedings of SIGMOD-2002*, June 2002.
- [9] U. Manber. Finding Similar Files in a Large File System. In *Proceedings of USENIX-1994*, January 1994.