A Preliminary Study of Size Optimization for Text-Based Web-Resource

Shinsuke Matsumoto
Graduate School of Information Science and Technology,
Osaka University
1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan
Email: shinsuke@ist.osaka-u.ac.jp

Masahide Nakamura
Graduate School of System Informatics,
Kobe University
1-1, Rokkodai, Nada, Kobe, Hyogo, 657-8501, Japan

***Abstract*—This paper discusses about file size optimization of text-based Web resources with the aim of network traffic reduction. From the perspective of network traffic reduction, wasteful representations written in text-based resources (e.g., indent, line break and comment) should be eliminated by applying any optimization techniques before the deployment. However, there are no common-sense of size optimization for text-based resources compared to multimedia resources. Our long-term goal of this research is to create or develop a literacy of size optimization for text-based resources on the Web. In this paper, we organize some existing size optimization techniques with these advantages and disadvantages. Next we conduct an experiment to show the effect of two optimization techniques using three famous JavaScript libraries. Finally, we discuss requirements and measures of server-side size optimization plugin that keeps both continuous of development and code openness of the Web.**

I. Introduction

This paper discusses file size optimization of text-based Web resources with the aim of network traffic reduction. The current Web is composed of a variety of Web resources. They can be broadly classified into two categories: multimedia resources (e.g., image, audio and movie) and text-based resources (e.g., HTML, CSS and JavaScript).

A size optimization for the *multimedia resources* is already widespread for most Web engineers because every multimedia resource is basically have a large data size and occupies network traffic. For the multimedia resources, these size optimization is generally called video/audio/image coding or compression. Applying these techniques are decided by the trade-off of its quality and size. Web engineers select loss-less format (e.g., png) or lossy format (e.g., jpeg, gif, mp4, mp3) according to features of optimized resource. The remaining challenges in the view of traffic reduction of multimedia resources are on the field of data encoding.

In contrast to that, applying an optimization technique for text-based Web resources is still unfamiliar compared to the multimedia resources in spite of there are many optimization tools. Optimizing text-based resources can affect to reduce overall Web traffic because of the Web (or REST) architecture called *code on demand*[1]. In this architecture, a server transfers a set of *raw source codes* (i.e., text-based resources) and a client executes itself on-demand. Therefore, preliminary size optimization by a compiler is not necessarily applied contrary to an architecture of binary (or multimedia) data distribution. For example, code representations, which are written for improving readability, such as indent, line break and comment are completely unnecessary for the execution on a browser. In addition, dead codes[2] such as unreachable code, unused variable and debugging code statement are also not necessarily eliminated.

From the perspective of reduction of network traffic, these wasteful representations should be eliminated by applying any optimization techniques before the deployment. Ihm and Pai pointed out that the size composition of multimedia resources takes over 50% in entire Web pages from the United States[3]. However, these multimedia resources have already optimized or compressed. On the other hand, although text-based resources takes about 30%, they might have a high potential for saving Web traffic. Especially the size of JavaScript file shall become larger with the future growth of HTML5 and client-side scripting technologies.

Our long-term goal of this research is to create or develop a common sense of size optimization for text-based resources on the Web. In order to achieve the goal, we are needed to show the effect of applying size optimization for the current entire web data. Furthermore, it is necessary to provide a tool or a server plugin that automatically applies some optimization techniques to a set of deployed text-based resources based on an user-specified optimization configuration. In this paper, first, we organize some existing size optimization techniques with these advantages and disadvantages. Next we conduct an experiment to show the effect of two optimization techniques using three famous JavaScript libraries. Finally, we discuss requirements and measures of server-side size optimization plugin that keeps both continuous of development and code openness of the Web.

II. Size Optimization Techniques of Text-Based Web Resources

Many optimization techniques have already been proposed and developed on the Web. In this section, we classify these techniques into the following six categories and discuss their features and side effects.

- Elimination of non-syntactical representations
- Abbreviation of syntactical representations

978-1-5090-1851-2/16 $31.00 © 2016 IEEE
DOI 10.1109/IWESEP.2016.16

- Optimization of syntax tree
- Dynamic self decompression
- Combined resources
- HTTP compression

A. Elimination of non-syntactical representations

This technique eliminates some non-syntactical representations that include indent, comment, line-break and unnecessary white-space[4][5]. This technique is a most simple and intuitive approach for minifying text-based resources. We introduce a concrete example in follows. Here is a JavaScript code which calculates and displays a result of sum of 1 through 10.

```
// initialization
var sum = 0;
// summation
for (var i = 0; i <= 10; i++) {
 sum += i;
}
// output
alert(sum);
```

We can achieve the following code by applying the non-syntactical elimination.

```
var sum=0;for(var i=0;i<=10;i++){sum+=i;}alert(sum);
```

The effect of code size reduction is about 50% (102 bytes to be 52 bytes). This technique may have high effects not only for JavaScript but also for HTML and CSS because they basically include a lot of indents and empty lines. The advantages of this technique are high intuitiveness, high applicability and few side effects to execution. Almost raw source code can be quite minified by this technique. The disadvantage is decreasing readability.

B. Abbreviation of syntactical representations

This technique abbreviates some syntactical representations such as variable name, function name and removable semicolon to shortest possible[4][5]. A sample code described in Section II-A is converted as below.

```
var s=0;for(var i=0;i<=10;i++){s+=i}alert(s)
```

A variable `sum` is abbreviated to `s` and two semicolons written in summing statement and alert statement are eliminated. The size reduction effect of this technique is 57% (102 bytes to be 44 bytes). In addition, the following tricky code, called condensed code[6], can be used in JavaScript.

```
var a = void 0;    // var a = undefined;
var b = !1;        // var b = false;
var c = {};        // var c = new Object();
var d = [];        // var d = new Array();
```

This technique is also effective for HTML and CSS. In HTML, some tag attributes such as id name or class name can be abbreviated. CSS supports shorthand properties[7] which abbreviate a color code or specifies many properties simultaneously. The following raw CSS code:

```
#container {
  margin: 0px 0px 0px 0px;
  color: #EEEEEE;
  font-style: italic;
  font-size: 1.2em;
  font-family: Arial, sans-serif;
}
```

can be shortened to the following

```
#container {
  margin: 0;
  color: #EEE;
  font: italic 1.2em Arial, sans-serif;
}
```

C. Optimization of syntax tree

This technique applies code size optimization of logical structure of the syntax tree. Examples are: elimination of unused variable declaration, elimination of unreachable code, elimination of redundant code, simultanous variable assignment and optimization of program flow. Basically, these techniques are applied for performance optimizion such as memory size assignment and execution speed. In the code on demand architecture, a size of source code can be reduced by applying these performance optimization.

An example code, described in Section II-A, is minified as follows by this optimization.

```
alert(55)
```

The most of popular JavaScript libraries including jQuery and bootstrap are already optimized before release to CDN server. These optimized libraries are basically named as `*.min.js`. We argue that size optimization of JavaScript libraries is already well developed.

D. Dynamic self decompression

In this approach, subject source code is regarded as just a string and compressed by Huffman coding or LZW before the deployment. Then, the compressed data is deployed with decompression code and a client decompresses dynamically[8]. Concrete procedure is follows.

1) A developer generates compressed string data $JS_{compressed}$ from a raw source code JS_{raw}.
2) The developer combines a compression function $decode()$ and $JS_{compressed}$.
3) The developer embeds the combined JavaScript data to a HTML file and deploys it on a server.
4) A client sends a HTTP request to the server and retrieves web resources.
5) The client executes $decode(JS_{compressed})$ dynamically, generates JS_{raw} and executes it by $eval()$ function.

We show an example of source code. The following JavaScript code includes a compressed string of a raw code, described in II-A, and decompression function $decode()$.

```
var decode = function(a) {
  var d = ...; // huffman decoding
  return d;
```

```
}
var compressed='1B1b1e1f2J2Y1c3K3_1I1X1[1]3
V3a1Y2R2S 1P1S1T3O3P1Q2[2g1L1M3f3h1J3Z3e1E1F3]3^1
C2h3 Y181;1>3'1<2Z3X19242U163S142f3W';
eval(decode(compressed));
```

This technique builds on "code on demand" architecture which transfers a script code as a string data to a client and execute on the client. Loss-less compression for source code data may have high potential because source code characters have high-frequency bias because of the reserved words (e.g., `for` and `if`). Furthermore, this technique allows combined usage of the previous optimization techniques.

The disadvantages are on trade-off between size of raw code and size of $decode()$ function and on execution performance. If a raw code is too short compared to a length of $decode()$ function, the deployed code should be larger. In the case of above example, the size of code is five times larger than raw (102 bytes to be 534 bytes). Additionally, a client is required extra execution time to decompression.

E. Combined resource

Some HTTP requests and declarations of relationship of multiple resources (e.g., `src=` and `rel=`) can be reduced by combining many text-based resources into a few number of files The following code which uses four libraries;

```
<script src="jquery.min.js">
<script src="jquery-ui.min.js">
<script src="jquery-mobile.min.js">
<script src="application.js">
```

is minified as follows by combining every libraries into a single "unified.js".

```
<script src="unified.js">
```

Although the size reduction effect is small, it has little side-effect for client execution.

F. HTTP compression

HTTP compression is an optional feature of HTTP/1.1 [9] to reduce the amount of HTTP traffic of Web contents [10]. It allows Web contents to be compressed on server-side before transferring to a client. In general, gzip format, which uses Lempel-Ziv (LZ77) algorithm with Huffman's coding, is used as a compression algorithm.

An example of a sequence of negotiation is follows. First, a client sends a request message to a server with "Accept-Encoding" parameter in the request header to enable HTTP compression.

```
GET /index.html HTTP/1.1
Host: example.com
Acccept-Encoding: gzip
```

Then, the server sends a response message with the following response header.

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
```

In this case, the requested and transferred HTML content (i.e., http://example.com/index.html) written in the response body is compressed by gzip. It is necessary for the client to decompress the response body before parsing the HTML content.

This technique is supported by a common Web server software (e.g., Apache HTTP Server, Internet Information Services and Nginx). However, the technique is disabled by default because it takes a minor performance hit on both client and server for the decompression. We consider that this disablement can be potential for size reduction of JavaScript resources.

III. Comparison of Size Optimization Techniques

A. Qualitative comparison

Table I shows comparison of six size optimization techniques described in Section II-A to II-F. All optimization techniques, except for HTTP compression, directly modify text-based resources themselves. Therefore, they have a disadvantage of decrease of code readability. Especially, optimized resources, which are applied optimization of syntax tree or dynamic self decompression, are almost impossible to understand without decoding. This reduction of readability may lose concept of *openness of the Web* which allows everyone to make copy or reference every Web resources.

HTTP compression and dynamic self decompression cause a decrease in execution performance of a client because of their extra process. Especially, dynamic self decompression may have low practicality because it drastically increases execution time and decreases code readability. On the other hand, the current Web servers enable to use HTTP compression. The current client devices have enough performance to decompress the compressed HTTP data in an ignorable short time.

We conclude that almost size optimization techniques have no adverse effects except decrease of code readability. Web developers should positively consider to use these optimizations. One of the important concerns for practical use of size optimization is how to avoid the decrease of readability.

B. Quantitative comparison

We have conducted an experiment in order to confirm the reduction effects of size optimization techniques. The experimental objects are famous and widely used JavaScript libraries.

- jQuery (ver. 2.1.1)
- prototype.js (ver. 1.7.2)
- backbone.js (ver.1.1.2)

Although the three libraries are provided with an already-minified version on their CDN servers (basically they are suffixed with ".min.js"), we use non-minified version. Subject optimization techniques includes HTTP compression (II-F) and YUI tool[11]. YUI is a minification tool that can process simple code optimization which includes both of elimination of non-syntactical representations (II-A) and abbreviation of syntactical representations (II-B). The reasons why we excluded other optimization techniques are: optimization of

TABLE I
COMPARISON OF SIZE OPTIMIZATION TECHNIQUES FOR TEXT-BASED WEB RESOURCES.

Technique	Subject			Side effect on client
	HTML	CSS	JavaScript	
Elimination of non-syntactical representations	✓	✓	✓	reduce readability
Abbreviation of syntactical representations	✓	✓	✓	reduce readability
Optimization of syntax tree	n/a	n/a	✓	reduce readability, improve performance
Dynamic self decompression	n/a[1]	n/a[1]	✓	reduce readability, increase execution time
Combined resources	✓	✓	✓	reduce readability
HTTP compression	✓[2]	✓[2]	✓[2]	increase execution time

[1] Basically, this method can not be directly applied to HTML or CSS file. However, if they are written in JavaScript and are reconstructed by JavaScript dynamically, the technique can be applied.

[2] This technique can be applied to all Web resources because it compresses the subject data on HTTP protocol level. However, it has less effect for binary or multimedia resources.

TABLE II
EFFECTS OF SIZE OPTIMIZATION FOR POPULAR JAVASCRIPT LIBRARIES

Subject library	Raw size	YUI[1]	HTTP compression	Both of YUI and HTTP compr.
jQuery	241.6 KB	128.2 KB (47%)	72.9 KB (70%)	37.0 KB (85%)
prototype	193.1 KB	102.3 KB (47%)	45.3 KB (77%)	33.0 KB (83%)
backbone.js	59.6 KB	19.7 KB (67%)	17.3 KB (71%)	6.9 KB (88%)

Bracket represents size reduction rate compared with raw file size

[1] Applied YUI compressor[11]. This tool enables elimination of non-syntactical representations (II-A) and abbreviation of syntactical representations (II-B).

syntax tree (II-C) has strong constraints for subject code to use it, dynamic self decompression is tricky and slightly unpractical, and combined resources (II-E) may have small effects compared with others,

Table II shows the result of the experiment. Percentage in parentheses represents reduction rate. Script sizes are reduced about 30% to 50% for each JavaScript library by applying YUI tool. In other words, applying simple code optimization can reduce the JavaScript size to 50% to 70%. HTTP compression also have high reduction rate about 70%. This is because Huffman coding provides effective performance for a text-based file. Moreover, we can achieve 90% reduction rate by using both optimization techniques. The reason is that the processing result of code optimization is just a text (JavaScript) file and HTTP compression is effective to a text-based file. Therefore, both optimization can independently contribute to size reduction.

The three JavaScript libraries have already been published optimized file named `*.min.js`. Furthermore, their CDN servers have also enable HTTP compression. We can argue that developers of these famous libraries are contributing internet traffic reduction by using various size optimization techniques.

IV. TOWARDS AUTOMATIC AND CONTINUOUS SIZE OPTIMIZATION

In this section, we introduce three existing optimization tools and discuss behavior of our proposed tool which provides automatic and continous size optimization.

A. Existing tools

There have been provided many optimization tools and services.

- **YUI Compressor**[11] is a well-known tool for JavaScript minification. YUI supports elimination of non-syntactical representations (e.g., omitting white-spaces, indents, and line breaks) and abbreviation of syntactical representations (e.g., shrinking variable names). The advantage of this tool is less side effects on script behavior compared with other minification tools.
- **Closure Compiler**[12] is one of an optimization tools provided by Google. In contrast to YUI Compressor, Closure Compiler supports powerful code optimization techniques. For example, the shortest optimized code described at Section II-C is processed by this tool. However, the tool imposes some restrictions on an optimized script in order to retrieve a maximally-optimized code.
- **packer**[8] is described as an obfuscation tool of JavaScript. However, in our classification, this tool can be regarded as a tool for applying dynamic self decompression. Firstly, the tool assumes subject code as just a string data and applies loss-less compression using Base62 encoding.

B. Idea of our proposed tool

Existing tools described in the previous section can be used freely through the Web. However, it is difficult to use continuously because subject source code need to be input by a developer manually. Additionally, deployment of optimized code may go against the concept of openness of the Web. The optimized code is hard to understand and will lead to less reusability. The current Web allows us to see and learn many interesting code snippets written from raw source code.

We assume that our objective size optimization tool is rquired to fulfill the following requirements.

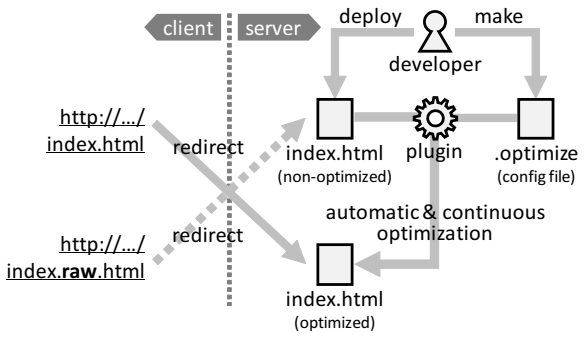

Fig. 1. Processing flow of size optimization plugin

- **R1**: Providing automatic and customizable size optimization.
- **R2**: Keeping openness of the Web for clients.

In order to meet the R1, optimization tool should be provided as a server-side plugin. A developer create a file for optimization customization like `.htaccess` file that is a configuration file for access control of Web resources. When a developer set the customization file (e.g., `.optimization`) to the subject resource directory, a server applies specified optimization techniques automatically.

To meet the R2, both of raw files and optimized files should be published on a server simultaneously. In this case, URI names of these two files must be different from each other. For example, a file name, `index.html`, is already a de-facto standard of welcome file name. Almost modern browsers complement `index.html` when the accessed URI ends with slash. If an optimized file is named `index.min.html` and deployed on a server, this complementat of welcome file name is disabled.

Figure 1 shows behavior of our proposed size optimization plugin which meets both R1 and R2. First, a developer deploys a HTML file (index.html) and an optimization customization file (`.optimize`) to the directory on a Web server. The plugin automatically loads these file and creates optimized files as temporary. This automatic optimization meets R1. When a client sends a HTTP request to http://.../index.html, the server redirects to the optimized file. The client can retrieve the optimized file and save network traffic without any additional effort. An endpoint http://.../index.raw.html allows access to non-optimized file in order to meet the R2. If a user want to check raw source code, the user only has to specify `*.raw.html` or `*.raw.js`.

V. Discussion

There is an opposite approach to our aimed common sense development of size optimization. Google provides an option to enable Web data optimization in mobile version of Chrome[13]. If a user enables this option, every HTTP request are bypassed to a Google proxy server. Then, requested Web resource is optimized on the proxy server and returns to the client.

The advantages of common sense development compared with the Google proxy are follows.

- Avoiding collection of browse history by Google.
- Enabling flexible and customizable size optimization.
- Faster response because of needless of proxy.
- Enabling size optimization to SSL traffic data.

The follows are disadvantages.

- Need to be developed common sense of size optimization.
- Impossible to optimize entire Web data.

VI. Conclusion

In this paper, we discuss about size optimization of text-based Web resources with the aim of network traffic reduction. We conduct an experiment to show the effect of two optimization techniques using three famous JavaScript libraries. Then we discuss requirements and measures of server-side size optimization plugin that keeps both continuous of development and code openness of the Web.

One of the most important future works is to conduct a massive empirical study to confirm prevelance rate and size reduction effect using a large number of current Web pages. Additionally, we must discuss detailed description of an customization file `.optimization`. We have a plan to develop the server-side plugin described in Section IV-B.

Acknowledgment

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No.26280115, No.15H02701), Young Scientists (B) (No.26730155).

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.
[2] J. Knoop, O. Rüthing, and B. Steffen, *Partial dead code elimination*. ACM, 1994, vol. 29, no. 6.
[3] S. Ihm and V. S. Pai, "Towards understanding modern web traffic," in *Proc. Internet Measurement Conference*, 2011, pp. 295–312.
[4] S. Souders, *High Performance Web Sites: Essential Knowledge for Front-End Engineers*. O'reilly, 2007.
[5] G. Frederick and R. Lal, *Optimizing Mobile Markup, Beginning Smartphone Web Development*. Apress, 2009, pp. 213–238.
[6] SitePoint, "19+ javascript shorthand coding techniques," http://www.sitepoint.com/shorthand-javascript-techniques/, (last accessed at December 2015).
[7] MDN, "Shorthand properties," https://developer.mozilla.org/en-US/docs/Web/CSS/Shorthand_properties, (last accessed at December 2015).
[8] D. Edwards, "A javascript compressor. version 3.0," http://dean.edwards.name/packer/, (last accessed at December 2015).
[9] R. T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, *Hypertext Transfer Protocol HTTP/1.1*. RFC Editor, 1999.
[10] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of http/1.1, css1, and png," in *Proc. Conference on Applications, technologies, architectures, and protocols for computer communication*, vol. 27, no. 4, 1997, pp. 155–166.
[11] J. Lecomte, "YUI compressor," http://yui.github.io/yuicompressor/, (last accessed at December 2015).
[12] Google Inc., "Closure compiler," https://developers.google.com/closure/compiler/, (last accessed at December 2015).
[13] ——, "Data compression proxy," https://developer.chrome.com/multidevice/data-compression, (last accessed at December 2015).