Hyper Text Transfer Protocol Compression

Dr.Khalaf Khatatneh, Professor Dr. Ahmed Al-Jaber, and Asma'a M. Khtoom

***Abstract*—**This paper investigates HTTP post request compression approach. The most common way to create HTTP POST is via the submission of a FORM with the method set to post (form post).In this paper I plan to explore the ability of compressing POST request on the browser side. We analyses the performance of the connection when transfer a compressed POST request and the results compared with the situation where is no compression. The impact of compressing POST request were observed, it results in additional bandwidth savings and less transfer time for the requests, results shown that about 47% of total transfer time has been reduced when the request is compressed. We conclude from comparisons that compression of POST requests can help too much in the HTTP transactions which offer powerful, reliable, and affordable HTTP transmitting for Web servers.

***Keywords*—** HTTP, POST Request, Gzip Compression Algorithm, Transfer Time.

I. Introduction

THE During the past few decades the World Wide Web (WWW) has become the focus of an intense research activity, performed by both academic and industrial research centers; this activity is mainly aimed at developing efficient techniques to retrieve information over the web in less response time and less bandwidth consuming.

One of the interested techniques used for improving and enhancing the web is the compression; since there are fewer packets traversing the network from the server to the end user, application data gets delivered faster and the required pages are displayed quickly for the users, which make good impression on the site.

POST method is one of eight HTTP methods that indicating the desired action to be performed on the identified resource, it is considered as the basis of html forms. On a browser, the most common way to create a HTTP POST is via the submission of a FORM with which consume more bandwidth and require more time for the request to be transferred from the browser to the server.

Bandwidth and transfer time must be consumed in an effective way that makes the usage of them more efficient and powerful. To address these needs, we propose a way to compress HTTP POST requests by compressing all data within it message body.

We analyses the performance of the connection when transfer a compressed POST request and the result shows that compression for the POST requests can help too much in the HTTP transactions which offer powerful, reliable, and affordable HTTP transmitting for Web servers.

Dr.Khalaf Khatatneh, Professor Dr. Ahmed Al-Jaber, and Asma'a M. Khtoom are with Faculty of Graduate Studies, AL-Balqa Applied University, 1July, 2007. (email ID : E-mail: dr.khalaf@bau.edu.jo, Jahmad@bau .edu.jo, AsmaKhtoom@yahoo.com)

II. Related Work

HTTP Response Compression

Compression technology can provide dramatic the method set to POST (a form post). Usually, the form inputs that have been uploaded are too large before it is sent to the client browser and then client's browser decompresses and displays the content to the user.

HTTP compression is a recommendation of the HTTP 1.1 protocol specifications for improved page download time that is only now receiving the attention it merits. Its implementation was sketched out in HTTP/1.0 (RFC 1945) and completely specified by 1999 in HTTP/1.1[2]. Case studies conducted as early as 1997 by the WC3 proved the ability of compression to significantly reduce bandwidth usage and to improve Web performance.

Too many organizations and companies enjoy in this field and a lot of them study the benefit of web compression techniques, three independent studies -- two conducted by the WWW Consortium (W3C) and one conducted for the Mozilla organization -- highlight the benefits of HTTP compression [6].

The author in [7] demonstrates some of the quantitative benefits of response compression and conducted a test to determine the potential byte savings for a number of popular web sites. Work has also been undertaken in the area of delta encoding of HTML files [5].

GET Requests Compression

At the client side- normally the browser-different ideas about compression are around the compressing of GET requests then multiplexing the corresponding responses as in [4]. In [3] a split proxy system for compression of HTTP transactions with mobile clients is proposed, where the author use the idea of compressing GET requests by using zlib version 1.1.3 algorithm but it doesn't take into account the compressing of POST request and how it effect the transfer of the http request.

III. POST Request Compression

In this paper, we plan to explore the ability of compressing POST request on the browser side, and as we see that different browsers don't support this type of compression so we propose

browser that creates application performance improvements over the HTTP, by compressing every html page in server side The compressed request is created and a special header field called "Is-Compressed" is added as new header field to give an indication about compressed request, if it set to "true" then server decompress the request and manipulate the data and if it set to "false" the server get uncompressed request.

A. Work Environment

POST compression software (Proposed Browser) was implemented using Microsoft Visual Studio.NET 2005, VB.NET code language, with .NET framework 2.0.

The implemented software (browser that creates the POST request) is running on Windows XP, service back 2, 1.3 GHz speed and 1 GB of RAM PC. After publishing the demo page, it is transferred from browser to the server with leased line connection of 1 Mb/s transfer rate and actual transfer rate equal to 100Kb/s.

B. Proposed Solution

I. Proposed Web Browser

Firstly, web browser that acts as a functionality of any other web browser such as IE browser is created.

Secondly, POST request class is created; this class allows the proposed browser to send an HTTP POST request to the server, that maybe compressed and maybe not, it depends on user demand; thus the user can allow the compression of its request and can disable it by using "Is-Compressed" header filed and then the server respond with appropriate response.

This header field is added as new POST request header fields because already headers in this request don't enable the compression of it. At this stage we ensure that data have been sent successfully to server side.

II. Server Decompression Process

Gzip decompression code is implemented on the compressed POST and implement decompression method in server side. false accept the request as it is without applying the decompression process.

III. Gzip Compression Algorithm

The compression was performed using Gzip algorithm which uses a compression strategy based on LZ77 and Huffman coding, and supports a streaming interface; level 9 compression (the best) was used with a test where the compression ratio and compressed file size are most important than speed. Level 1 (the fastest) used with a consideration of the speed factor. The full story about how this algorithm work can be found in [1].

IV. Results

Different files are uploaded with aspx form page that published over the internet on (WWW.Mr Ayyash.Com) web site, it consists of: three text fields, one field for uploading files and submission button. This form and it is data are sent via POST request to the server, with one uploaded file at a time. The POST is sent into two modes: compressed mode and non compressed mode according to "Is-Compressed" header field.

The average values of transfer time, compression ratio and compression /decompression times for each request are computed after sending it many times via leased line medium of 1 Mb/s speed. 50 POST requests were sent to the server, each request was sent 10 times and the average of each factor is computed, two runs were done, one for level 1 and another for level 9.

Transfer Time Measurements

The time needed from the moment of sending the request until receiving the response. Results shown that about 47% of total transfer time has been reduced when the request is compressed and according to this reduction more bandwidth will be available to the users.

CRC values stand of Cyclic Redundancy Checker it is computed by the algorithm for each request before compression and after compression to indicate that it is server side to allow it receive compressed POST request and checks "Is-Compressed" header field if it set to true then decompress the request and if it is received correctly by the server which decompresses it.

TABLE 4.1
TRANSFER TIME FOR MULTIPLE REQUESTS OF 2GZIP LEVELS

Gzip Level		**Level 1(Fast)**		**Level 9 (Best)**	
Req .#	**Trans. T. [sec]**	**CRC Before**	**Trans. T. C. [sec]**	**Trans. T. C. [sec]**	**CRC After**
1	23.43	'744ED716'	21.92	21.99	'744ED716'
2	12.04	'689EC767'	9.02	8.89	'689EC767'
3	51.82	'8F1D6A41'	25.57	24.52	'8F1D6A41'
4	2.65	'ABCBF199'	1.8	1.66	'ABCBF199'
5	2.47	'3D779906'	2.56	1.87	'3D779906'
6	4.19	'3889322B'	2.49	2.42	'3889322B'
7	6.58	'3E0ACA9D'	3.31	3.29	'3E0ACA9D'
8	25.84	'9678A2EA'	18.34	16.56	'9678A2EA'
9	35.78	'C9D6F27A'	12.04	10.21	'C9D6F27A'
10	2.95	'4CB7AAFD'	1.75	1.66	'4CB7AAFD'
11	27.51	'189931E4'	13.75	12.9	'189931E4'
12	14.97	'8A0D1593'	7.61	6.92	'8A0D1593'
13	7.49	'E97B6133'	2.23	2.09	'E97B6133'
14	3.75	'EE1336A8'	1.85	1.83	'EE1336A8'
15	3.1	'46AF3A1F'	3.16	4.39	'46AF3A1F'
16	24.9	'CD6495E2'	24.87	25.07	'CD6495E2'
17	0.94	'D48B7EFF'	0.82	0.81	'D48B7EFF'
18	29.99	'B9ECF888'	29.71	29.81	'B9ECF888'
19	27.37	'9100551D'	27.52	28.19	'9100551D'
20	0.87	'159C1E0A'	0.82	0.81	'159C1E0A'

Where:
- Trans. T. [sec]: Transfer time without compression in seconds.
- Trans. T. C. [sec]: Transfer time with compression in seconds.

CRC: Cyclic Redundancy Checker

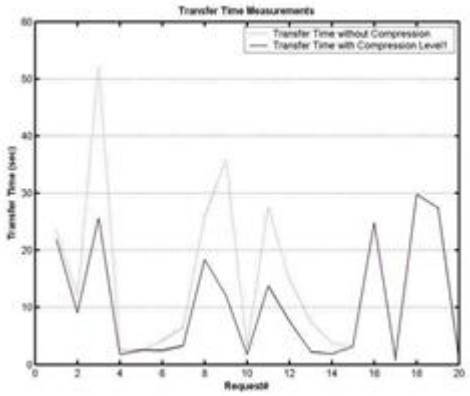

Fig. 4.1.a: Transfer time with and without compression versus request# - Level 1

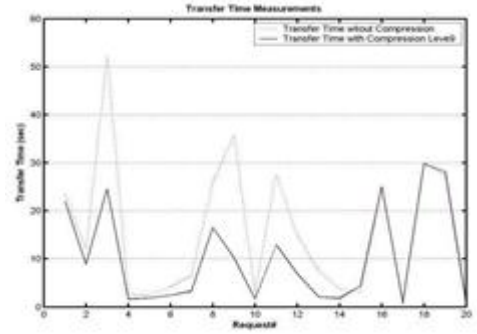

Fig. 4.1.b: Transfer time with and without compression versus request# - Level 9

Compression Ratio Measurements

This value shows the differentiation on the request size before (Original size) and after compression (Compressed request size). It is given in the following equation:

$$CR = 1 - \left(\frac{CS}{OS}\right) * 100\%$$

Where
- CR: Compression Ratio.
- CS: Compressed request Size in KB.
- OS: Original request Size in KB.

The results in table 4.2 and Figure 4.2 show that compression ratio (CR) differ from gzip level1 (fastest) which give less compression ratios from level 9 (Best).

TABLE 4.2
COMPRESSION RATIOS FOR MULTIPLE REQUESTS OF 2 GZIP LEVELS

Request #	Original Size [KB]	Level1-CR	Level9-CR
1	800.9199	7.32	8.08
2	380.4961	26.37	27.42
3	1864.89	53.07	54.87
4	36	72.29	73.36
5	32	69.03	69.51
6	92	62.89	64.51
7	172	61.53	64.39
8	1015.803	40.54	45.03
9	1228.834	67.87	74.89
10	57.5	77.86	79.32
11	964	52.87	55.79
12	498.5	53.69	59
13	224	87.56	88.88
14	86.5	79.16	80.65
15	60.89551	4.04	4.68
16	861.9043	0.11	0.11
17	1.181641	62.81	64.71
18	1053.601	1.63	1.38
19	963.2314	1.1	1.01
20	0.772461	47.03	47.54

Figure 4.2 is results in less compression ratio which means less file size and this lead to less transfer time.

Compression/Decompression Time Measurements

A. Compression Time Measurements

The time required to compress POST request which normally consist of text fields and uploaded files in second's 'sec'. The values of Compression time are computed on the client side (proposed browser) as shown in table 4.3 and figure 4.3.a these values are higher in case of level 1(fastest) than level 9 (maximum) this refer to how the algorithm work.

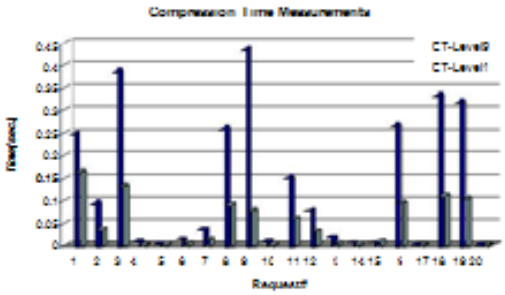

Fig. 4.3.a: Compression time versus request#

TABLE 4.3
COMPRESSION/DECOMPRESSION TIMES OF POST REQUESTS FOR 2 GZIP LEVELS

Gzip Level	Level 1		Level 9	
Request #	Comp. Time [sec]	Decomp. Time [sec]	Comp. Time [sec]	Decomp. Time [sec]
1	0.16204	0.02633	0.24634	0.02754
2	0.03435	0.01482	0.09231	0.01087
3	0.12884	0.04982	0.38527	0.04448
4	0.0021	0.00063	0.00822	0.00058
5	0.00494	0.00057	0.003	0.00056
6	0.00638	0.00198	0.01447	0.00184
7	0.01365	0.00381	0.03457	0.00334
8	0.08851	0.02822	0.25675	0.02573
9	0.07617	0.02316	0.43421	0.01723
10	0.00294	0.0109	0.00841	0.00078
11	0.06031	0.03459	0.14789	0.01991
12	0.03242	0.02263	0.07575	0.00935
13	0.00699	0.0019	0.01779	0.00166
14	0.00378	0.00113	0.00714	0.00099
15	0.00881	0.00183	0.00726	0.00354
16	0.09447	0.03321	0.26408	0.00701
17	0.00017	0.00004	0.00045	0.00004
18	0.10956	0.03731	0.32961	0.03707
19	0.1017	0.03417	0.31429	0.03598
20	0.00024	0.00004	0.00057	0.00004

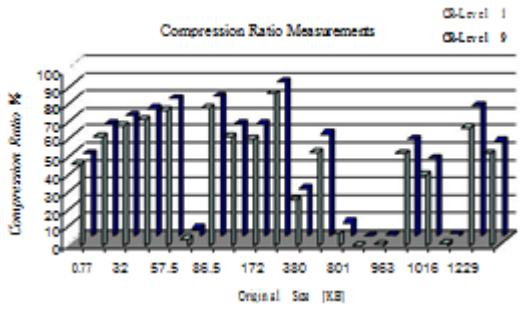

Fig. 4.2 Compression ratio versus original request size

B. *Decompression Time Measurements*

The time required to decompress POST request in seconds.

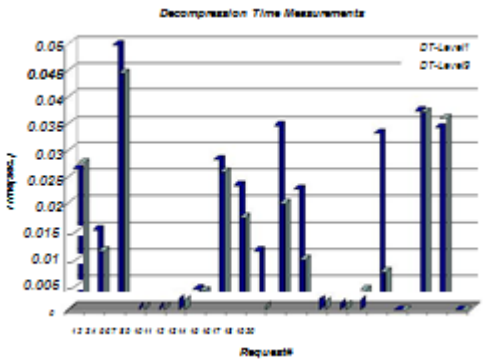

Fig. 4.3.b Decompression time versus request#

Nearly, decompression time is similar for both levels and is computed on the server side. It is the time consumed by the server to decompress the compressed request it received.

V. CONCLUSIONS & FUTURE WORK

In general, POST request compression is an effective way to achieve too many benefits in data transferring through the web, from this research we found it is very logical and useful to compress POST requests, especially, when we have large uploaded files, low speed network and high network traffic. POST compression results in additional bandwidth savings and less transfer time for the requests.

Future works can be done into two fields:

Firstly, there are many file types such as .jpeg, .gif images, .rar and .gz files where it is efficient to exclude them from compression process. hence, they are already in compressed format and do not compress much further; certainly no discernible performance is gained by another incremental compression of these files.

Secondly, depending on the content type of uploaded file in specific POST request, apply the appropriate compression algorithm. This can be done by allow the browser to choose from different types of algorithms that are suitable for some types of files than another. Because as we see from the results that gzip is work in efficient way for text files, it gives higher compression ratios and less transfer time than PDF files.

VI. ACKNOWLEDGEMENTS

Abed Al-Qader Ayyash has contributed significantly to produce this work.

REFERENCES

[1] Deutsch, P. (1996). "GZIP file Format Specification version 4.3", RFC 1952.

[2] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee (1999). "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068.

[3] Krashinsky, R. (2000). "Efficient Web Browsing for Mobile Clients using HTTP Compression", Distributed Operating Systems (6.894) Term Project MIT Laboratory for Computer Science, Cambridge, MA 02139.

[4] Mattson, R. and S. Ghosh (2005)." HTTP-MPLEX: AN APPLICATION LAYER MULTIPLEXING PROTOCOL FOR THE HYPERTEXT TRANSFER PROTOCOL (HTTP)", La Trobe University, Melbourne, Australia.

[5] Mogul, J. C., Douglis, E., Feldmann, A. and B. Krishnamurthy (1997). Potential benefits of delta-encoding and data compression for HTTP. In M. Steenstrup, editor, New York. ACM Press.
[6] Nielsen, H., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Wium Lie, H. and C. Lilley (1997). "Network Performance Effects of HTTP/1.1, CSS1, and PNG" NOTE.
[7] Timothy J. McLaughlin (2002)."The Benefits and Drawbacks of HTTP Compression", Department of Computer Science and Engineering, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015