Web Service Improvement Model Based on XDrill Algorithm

YANG Yan, YAO Hua-Xiong, LI Rong

Computer School, Central China Normal University, Wuhan 430079, China
Email：yanyang_wh@sina.com

Keywords: Web services; SOAP; compression algorithm; XDrill

Abstract: We will firstly apply XDrill compression algorithm to the SOAP protocol. Firstly, this paper makes in-depth analysis of the differences in principles between XDrill algorithm and XMill algorithm, pointing out the underlying causes for higher compression efficiency of XDrill, and confirms this view by a large number of experimental data. Then, XDrill algorithm framework as well as algorithm implementation suitable for Web services are proposed. Finally, we further validate the efficiency of algorithm model after optimization through specific experiments. On the other hand, we conduct valid document compression for SOAP protocol, reducing the Web service transmission times. Through a large number of experimental data, we find that the response time under XDrill-SOAP protocol is reduced by about 50% on average than traditional Pure-SOAP protocol, with an average reduction in the response time of about 10 % compared with XMill-SOAP protocol.

Introduction

The distributed technology of Web services is widely used in the various fields of computer. Usual XML data redundancy is relatively large, which increases SOAP document volume, and excessive network bandwidth is occupyed in transmission, thereby affecting the transmission speed of data [5]. If the SOAP document can be compressed, it can reduce the network transmission data volume and users' response time. Traditional compression algorithms include Gzip and XMill, able to compress data to 40% and 50% of its original size [6] in the case of completely saved data information. Among them, XMill algorithm is widely used and is the current mainstream SOAP compression algorithm. Although this algorithm can effectively complete single document compression, compression of documentation set is inefficient [7]. Gzip algorithm also has the same problem.

XDrill algorithm

Web service invocation

Credibility in this paper is mainly calculated based on trusted context that users are concerned, and the trusted context is indicated by ontology; conceptual match of context that users are concerned and the context of service provider, and the match result is an important basis for valuation of credibility, which can be used to adjust the weight of trusted contexts in the calculation of credibility. Because the trusted context is mainly described based on the concepts in the ontology, the matching algorithm is achieved by subsumption reasoning of concept. Supposed that the concept of trusted context that the service requestor is concerned is described as A, and the context of the corresponding trusted context for service provider is described as B, then the matching algorithm can be expressed in Table 1.

If the described pair of trusted context is of the same concept or equivalent concepts ($A \equiv B$), they are called complete matching. When a pair of trusted context is a complete matching, the weight for

trusted context is not adjusted, that is, adjustment factor of 1. If there is $A \supset B$, they are called tight match. Because at this time, for any instance a, if a∈B is satisfied, a∈A must be established, and that is, the context of service providers can meet the needs of context for the service requestors. That is, when the concept level of B is lower than A, the context described in B is more specific, able to meet the needs of its abstract context. For example, when the requester's need of location context is "Beijing", and the corresponding location context of service provider is "Fangshan" (Beijing ⊃ Fangshan), they are clearly in line with the requirements. When the trusted context is tight match, adjustment factor for the weight of trusted context is set to 0.95. If there is $A \subset B$, they are called general matching. Because in this case, the conceptual level of requestors' demand for context is described lower, and the context concept corresponding to the service provider is more abstract, and it is either possible or impossible to meet the context needs of the requestor. When the trusted context is generally matched, adjustment factor for the weight of trusted context is set to 0.5. If there is $\neg((A \equiv B) \vee (A \subset B) \vee (A \supset B))$, that is, it is described that there is no connection of concepts between a pair of trusted context, it can be determined that they are unmatched. In this case, adjustment factor for the weight of trusted context is set to zero.

Information redundancy of XML document set

We know that when users view the financial market information, they often need to look over information for several days or several weeks; these information are the same in structure but are slightly different in the data, and we usually call these XML document as document set. Because of the large data amount of financial information, and meanwhile different documents are extremely similar in structure and content, simple and storage usually will generate a lot of redundant information and occupy a lot of network bandwidth. If the compression of updated data can be conducted using the already compressed information, it will significantly reduce bandwidth consumption, and in particular, the use of incremental storage can effectively store the entire document set. The specific approach is to store one of several versions completely, and the others are stored in the form of incremental script [8]. If we can achieve incremental storage and updated operations for documents, the size of the document set after compression can be reduced. Thus, this approach can significantly reduce network bandwidth consumption compared with the conventional XML document compression.

Traditional XML document compression algorithms include XMill, XGrill, XPress, etc. and do not support incremental storage and update. Take XMill example, the core idea is to separate structure and data, and then to group and to store [9]. Usually, we call tags within XML document and attributes information as structure and their property values as data. XMill puts the tags and attribute information into dictionary with assigned labels; the property values are put into container, and common compression methods such as Zip are finally performed. Through this process, XMill usually will shrink XML document to 50 % of its size. However, there is a common drawback in XMill and other traditional algorithms that they are only able to complete the compression of single XML document. In the multi- document processing, these algorithms cannot get links between documents, unable to effectively deal with information redundancy of documentation set, thus it is difficult to improve the compression efficiency.

Different from traditional algorithms, XDrill will firstly divide XML document tree, and then tap the redundant information within the document and between documents according to the division results. Main features of XDrill algorithm are that it has fully considered the similarity in the structure of XML documents and obtained cluster groups by k-means clustering algorithm to calculate differences between various XML documents [10]. Specifically, XDrill algorithm has considered each document inside XML document set separately, and they are divided into segments and stored in accordance with incremental compression. Then, the difference values are calculated

based on these fragments so as to guide compression of subsequent fragments. For example, when the N-th fragment within XML document is compressed, the algorithm calculates the difference based on the N-th fragment inside the previous document for incremental storage compression.

Here, we use two specific experiments to compare the compression ratio of XDrill, XMill and Gzip algorithms in case of single document and multiple documents.

Experiment

Experimental environment: processor: Intel (R) Core (TM) 2 DUO CPU; RAM: 2.3G; master frequency: 2.2GHz; operating system: Window XP.

Experiment 1 Comparison in performance of XDrill, XMill and Gzip algorithms under single document

Table 1 Experimental Document Statistics

File name	（KB） Document Size (KB)	Repetitive structure	Whether the texts are regular
test	26	more	yes
file	40	less	no
tree	5	more	yes
blank	76	less	no

Experimental data: As shown in Table 1, we choose four XML documents in the literature, of which there are more repetitive structures in test.xml and tree.xml, while less repetitive structures in file.xml and blank.xml.

Experimental results: For the four data sets in Table 1, we use XDrill, XMill and Gzip algorithms for compression. To compare the difference between them, we define compression ratio: $1-\frac{File\ size\ beforecompression}{file\ size\ aftercompression}$. Higher compression rate indicates better compression effect of the algorithm.

Table 2 Comparison in compression efficiency under single document for three compression algorithms

File name	Gzip algorithm	Compression ratio XMill algorithm	XDrill algorithm
test	0.85	0.87	0.88
file	0.69	0.68	0.69
tree	0.93	0.95	0.96
blank	0.62	0.7	0.69

Table 2 shows the basic results. It can be seen that for a single document, three algorithms has close compression performance. Among them, there are more repetitive structures in test.xml documents and tree.xml documents, and then XDrill has slightly better compression effect than XMill and Gzip. blank.xml document is very irregular, and then XDrill has slightly worse

compression effect than XMill and Gzip. Overall, however, in the case of a single document, XDrill, XMill and Gzip algorithms have similar compression performance.

Experiment 2 Comparison in performance of algorithms under document set

Experimental data: In Experiment 1, four XML documents are taken as a basis and four document sets, namely, testO, fileO, treeO and blankO are established, and four document sets are composed of 1000 documents of test.xml, file.xml, tree.xml and blank.xml respectively.

Experimental results: For the document set, compression ratio of XDrill algorithm is significantly higher than XMill and Gzip algorithms.

Table 3 Comparison in compression efficiency under document set for three compression algorithms

File name	Gzip algorithm	Compression ratio XMill algorithm	XDrill algorithm
testO	0.54	0.59	0.75
fileO	0.44	0.42	0.57
treeO	0.62	0.66	0.83
blankO	0.37	0.41	0.55

Table 3 is comparison in compression efficiency under document set for three compression algorithms. With four kinds of different data, XDrill compression ratio has increased substantially, and compared with Gzip, the compression ratio is increased by an average of about 20 percent, and compared with XMill, compression ratio is increased by an average of 15%. Experiment 2 has verified XDrill advantage in multi-document compression.

Framework and implementation of XDrill algorithm

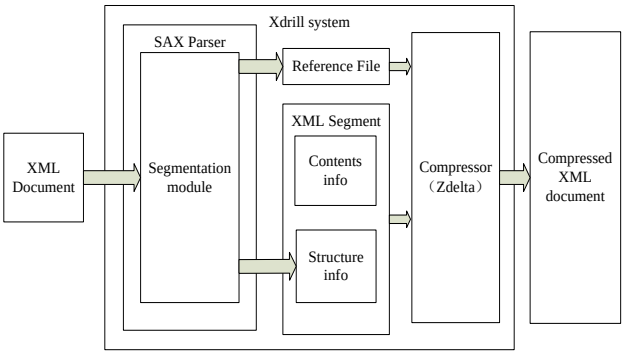

Figure 1 Frame structure of Xdrill compression system

Figure 1 is a framework organization chart of X Drill compression system algorithm, and XDrill compression system consists of two parts.

The first part is SAX parser, and the main function of this section is to read the original document and to invocate calls segmentation module, thereby generating corresponding reference and destination files. Another part is the compressor module, and its main function is to achieve compression of the original file by invocating zdeha compressor at bottom. Next, we discuss the basic realization of XDrill compression algorithm.

Algorithm 1. X Drill compression algorithm

Input: XML document collection

Output: compressed XML document collection

Variables: SAX- Event: SAX event

Description: Zdelta (refl, ref2, tar) is the compression function. Parameter refl and parameter ref2 are reference documents, and parameter tar is the destination file.

refl_structure,refl_contents,ref2_structure,ref2_contents,tar_structure,tar_contents Six variables in total, namely, refl_structure, refl_contents, ref2_structure, ref2_contents, tar_structure and tar_contents are buffer areas for system maintenance.

1: if (SAX-Event is the opening tag event)
2: The opening tag character is written into tar_structure;
3: else if (SAX-Event is the end tag event)
4: if (does not meet the division rules)
5: The end tag character is written into tar_structure;
6:else
7:zdelta(refl_structure,rer2_structure,tar_structure)
8:zdelta(ref1_contents,ref2_contents,tar_contents)
9: Move information of ref2_structure to ref_structure;
10: Move information of ref2_contents to ref_contents;
11: Move information of tar_structure to ref2_structure;
12: Remove the contents in tar_structure;
13: Move the information in tar_content to ref2_content;
14: Remove contents in tar_contents
15: else if (SAX-Event is text event)))
16: The symbol " ^ " is written into tar_structure;
17: The text content is written into tar_contents:
18: The symbol " ^" is written into tar_contents:
19:end

As can be seen from the algorithm flowchart, XDrill compression system has a total of six system caches, and each system cache has its own functions. Among them, function of refl_structure and ref2_structure is to store structural fragment information of the two reference files; function of refl_Structure and ref2_contents is to store content fragment information in two reference files, while function of tar_Structure and tar_contents is to maintain the current fragment information of XML document.

In the compression process, SAX parser reads the content from XML document and writes the content is in tar_Structure and tar_contents. If the current SAX event is the cutting breakpoint able to meet cutting rules, code on the 7th and 8th line will call zdelta compression tools and use structure and content fragments to compress destination file, and the code on the 9th-14th line means the data exchange in buffer area. Data in refl_structure and ref2_contents are updated using ref2_structure and ref2_contents. Similarly, the data in tar_structure and tar_contents are replaced with data in ref2_Structure and ref2_contents. Finally, tar_structure and ta_contents are emptied. SAX parser continues reading the following data. Decompression process threads are the same, while the process is just the opposite.

Common incremental compression algorithms include vcdiff, XDelta, zdelta etc., and zdelta is used as incremental compression algorithm herein. XDrill firstly locates fragment, and then conducts incremental compression using zdelta algorithm. For example, for the previous example, the user needs financial market information, and new XML documents only need to use the previous XML document as a reference document to achieve incremental compression, thereby greatly saving transfer time of data.

Conclusion

Through analysis on experimental data in the paper, XDrill algorithm has greater efficiency in compression of document set than today's conventional compression algorithms, and XDrill algorithm greatly improves the transmission speed of Web services so as to reduce the response time of the system. In addition, with increasingly greater transmission of document set, reduction in the system response time is more obvious. With the rise of mobile Internet, information transmission volume of Web services is in rapid growth. In this case, how to effectively use the features of mobile Internet to further improve the transmission speed of Web service and to reduce response time is a subject worthy of study.

Acknowledgements

the Humanity and Social Science Research Foundation of Ministry of Education of China under Grant No. 15YJA880095

Reference

[1]Y. Geng, J. Chen, K. Pahlavan, Motion detection using RF signals for the first responder in emergency operations: A PHASER project, 2013 IEEE 24nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC), London,Britain Sep. 2013

[2]Yishuang Geng, Kaveh Pahlavan, On the Accuracy of RF and Image Processing Based Hybrid Localization for Wireless Capsule Endoscopy, IEEE Wireless Communications and Networking Conference (WCNC), Mar. 2015

[3]Jie He, Yishuang Geng and Kaveh Pahlavan, Toward Accurate Human Tracking: Modelling Time-of-Arrival for Wireless Wearable Sensors in Multipath Environment, IEEE Sensor Journal, 14(11), 3996-4006, Nov. 2014

[4]Lv, Zhihan, Liangbing Feng, Haibo Li, and Shengzhong Feng. "Hand-free motion interaction on Google Glass." In SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications, p. 21. ACM, 2014.