Global Scientific JOURNALS

GSJ: Volume 7, Issue 10, October 2019, Online: ISSN 2320-9186
www.globalscientificjournal.com

ENHANCEMENT OF SOAP WEB SERVICES MODEL PERFORMANCE BASED ON OPTIMIZED XML FILES

Adeeb M. Al-qershi
Computer Center and Information Technology
adeeb@taiz.edu.ye

Abstract

The quality of Web Services (WS) depends on the efficiency of considered WS model. In a real environment, the efficiency of WS models are measured and evaluated by various vital metrics such as: reducing Response Time (RT), CPU Utilization and memory space. Traditional WS model has three components, which are WS Provider, WS Consumer and WS Registry as in [5],[3] .Actually, WS have two types based on principles either SOAP or REST as in [6].Each SOAP and REST has advantages and disadvantages over others. SAOP depend on XML files rather than REST that can operate on XML or JSON files. In fact, XML files have a large size and more manipulating time rather than JSON files. In this paper, proposed component has been added and tested into WS traditional model that has been called XMLOptimizer. Based on results of experiments that have done on WS model with XMLOptimizer, the XMLOptimizer enhanced the performance of WS. The measured factors of performance were Response Time (RT), CPU Utilities and Memory space. For instance, the RT of search operation has been enhanced to 99% as well as the memory space of XML files have been reduced by 90%.

Keywords: Web Services, SOAP, REST, XML, JSON, XMLOptimizer, Response Time, CPU Utilities, Memory space

Introduction

With the advent of the Internet, Web and e-commerce technology began to become as a commodity [1].WS technologies are becoming increasingly important for integrating systems and services [2]. They provide a universal and standard platform which can works with different services. The most common way to implement WS are SOAP and REST [4]. Based on table (1) each one them have advantage and disadvantage, but JSON files have less size compared by XML files. Consequently, XML files exhausted more memory space, CPU utilization and RT.

Literature Review

SOAP: " is a standard for sending messages and making remote procedure calls over the Internet"[11].so SOAP is independent of the programming language, object model, operating system and platform. It uses HTTP as the transport protocol and XML for data encoding"[11].

REST is Representational State Transfer and it depend on client and server architecture REST does not require message format as SOAP [7].

Table (2.1) demonstrates a summery comparison between SOAP and REST depend on evaluation of performance was performed on mobile emulator [12], [13], [14].

Table (1): SOAP and REST performance [12], [13], [14].

Factor	SOAP	REST
Architecture	Enterprise	Client/server
Designed for	Extensible, Distributed Computing	Client/server
Stateless	Yes	Yes
Message Format	XML	XML and JSON
Message size	Little big	Very little
Application	Heavyweight	lightweight
Mobile computing	Little suitable	More suitable
Coupling	Tightly coupled with Client-server but Loosely Coupled with enterprise	Opposite of SOAP
Standard	Standard	Ad-hoc
Response time	Relatively high	Low
Security	Secure (based on WS-Security)	Low secure
Transaction	Support ACID	Not
Message size	9-10 times bigger than REST	Smaller than SOAP
Latency response time	5-6 times more than REST	5-6 lesser than SAOP

There are many studies have been done to enhance WS model by reduce XML files. This reducing depended on compress XML files size. This technique applies either on sent SOAP message or XML files. There are many techniques are used to compress XML files to enhance XML file performance such as ZIP, XMILL, XGrind and XPRESS. These studies listed below.

1- ZIP:

(Tere, G.M., R R Mudholkar, and B.T. Jadhav,2014) have depended on their research on compress messages, which are sent from SOAP as a response files [8]. This compression used ZIP utilities. Its good way to send XML data from server side to client side, but it's not clear how will deal with this ZIP file in client side.

2- XMill

XMILL uses dictionary compress [9] to compress XML tags and zlib, which is library of gzip to compress inner data [10].

3-XGrind

XGrind depends on Huffman encoding to compress data and dictionary compressing for XML tags [9].

4-XPRESS

XPRESS technique based on the XPRESS compression rate was 73% and query performance was 2.38 times better than other XML compressors [9].In this technique six encoder's methods are done for data values u8, u16, u32 and f32, dict8 and Huffman.

The previous XML compression techniques still needed to compress and decompress of entire data every time to access it, so it's not effective and not flexible especially with the frequented accesses. Because, these techniques need additional time to manipulate access operation (compress time+ decompress time+ query time). Furthermore, decompress XML file will return XML file to its original size (before compress). Thus, each access to XML file needs additional memory space.

Theoretical framework

XML optimizer: is a component that has been developed to improve client side application performance based on reducing the XML file size and RT of operations on XML files. In addition, all operations on this file are done in compressed mode as shown in Figure (1).In fact, XMLOptimizer reconstruct and rearrange XML file tags and data in efficient way.

1- XML Optimizer Functionality

XML files have plenty of whitespaces, because each tag (element) takes a separate line, even if tag characters not fill all line, the remaining spaces are padded with whitespaces. *Moreover*, XML File has a lot of tags that are repeated for each record in both open and close tag. Figure (1) shows how XML Optimizer optimizes the original XML file by eliminating Whitespaces and reducing XML tag count of original XML file

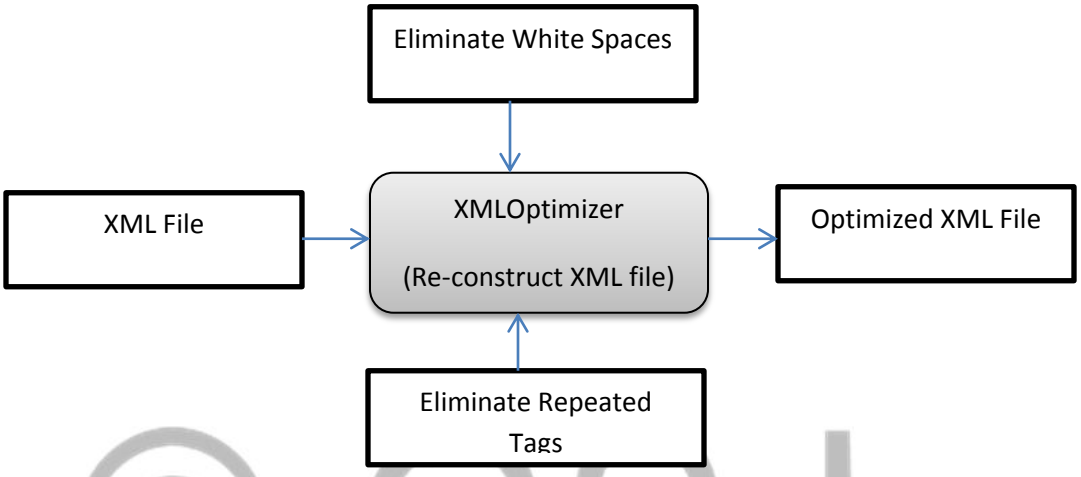


Figure (1): XML Optimizer Architecture

The XML Optimizer receives XML file from WS Provider, then it eliminates white spaces and repeated tags. The XML optimizer has two main stages to achieve optimizing operation. These steps are explained as follow:

First Stage:

XML optimizer in this stage holds the original file that is decompressed in first stage and collects all tags within original XML file without repeating. These tags are used to build the compressed xml file as well as used in the third stage to collect data from all tags. This stage reduces the number of tags in the compress xml file compared with original one by eliminating repeated tags. The next equation is used to calculate count of XML file tags to show the difference between tags count of original and compressed xml files sequentially. $(x) = (2 * R_t) + (2 * P_{tn}) + (2 * C_{tn})$

Where: R_t is Root tag, P_{tn} number of parent tags, C_{tn} number of Child tags, 2 constant number for open and close tag.

. This equation calculates tags count in traditional xml file. When applying this equation on xml file before compressing, the number of xml tags is 44 tags as in figure (2). *In the other side*, when applying the same equation on xml file after compressing, then the tags number is 16 tags as in figure (3).*Consequently*, the result of this stage is XML structure without repeated tags.

Second Stage:

In this stage the content (data) of all tags in original file is collected and then added to previous structure that has been generated in previous stage as in figure (2).So, XML Optimizer functionality is summarized in:

1. Received XML file.
2. Eliminates white spaces of XML file.
3. Eliminates repeated tags of XML file.
4. Generates new XML file structure.
5. Fill the generated XML structure with data.

The size of optimized XML file depend on XML tag length, number of tags and number of whitespaces ,so the longer length of XML tag ,the smaller optimized XML file size ,also the more whitespaces , the smaller optimized XML file size.

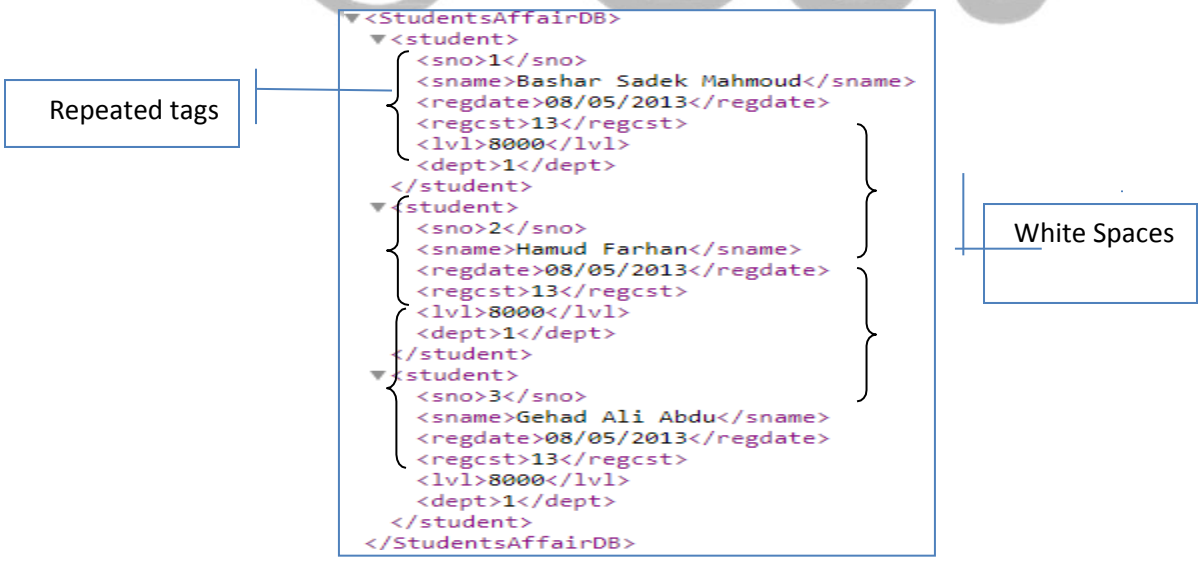


Figure (2): Traditional XML file

The xml file after XML Optimizer has been applied looks like as in figure (3). The repeated tags are eliminated and their data is collected and allocated within unique tags

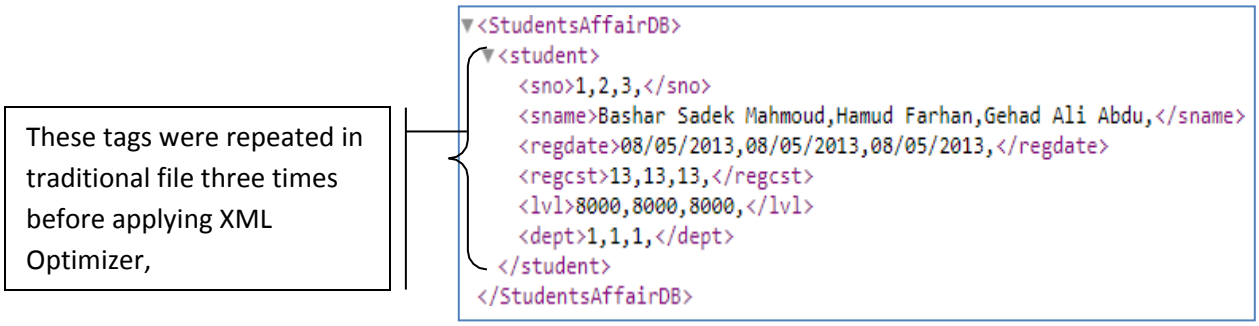


Figure (3): WSMM Compressed XML file

The figure (3) shows the xml file after compress the xml file .Xml Optimizer works with two main methods *XmlUniqueSructure* and *XMLOptimizer*.*XmlUniqueNodes* method is used to travel throw original XML file nodes (tags) to collect all nodes Root, Parents and Childs without repeating. These nodes used to build the structure of new compressed XML file. *XMLOptimizer* method is used to re-construct the XML file structure. The new structure contains the unique tags that are returned by *XmlUniqueNodes* method, and all data of XML file nodes. *XMLOptimizer* method travels all nodes to get data from each node and appends it into matches tag node.

2- XML Optimizer Contribution in WS

In the contrast of traditional compression techniques that depend on compressing and decompressing files, the XML Optimizer doesn't need to do that each time.

The idea behind XML optimizer is eliminating whitespaces as well as eliminating repeating tags of XML tags. *Consequently*, XML Optimizer contributed to:

1- No need to decompress xml files, so all operations are done on compressed xml file.

2- Save memory space due to reduce of XML file.

3- Enhance XML parser by reduce RT of operations on XML files such as search, update, insert and delete.

4- Save time of compress and decompress operation that is needed in others techniques.

5- Save CPU Utilities.

6- Works with multiple Tables in the same XML file.

7- Can apply with large size of XML files.

3- XML Optimizer Algorithms

Algorithm: *XmlUniqueSructure*

Input Data : origin XML file path

Result: new XML file structure without repeated tags

```
1  begin
2      define  XMLObj ← new XMLDocument
3      define  root ←string
4      define  parents [],childs [] ← XmlNodeList
5       XMLObj.load (xmlPath)
6      XMLObj.isExist (orginalFile)
7      if XMLObj =true then
8          parents=distinctParent( XMLObj.SelectNodes(root))
9         for each parent in parents loop
10            Childs[]= XMLObj.getChilds(parent)
11        end loop
12     end if
13 end
```

The previous pseudo code shows how parse received XML file to get Parents and their Childs into strings without repeating. In line 8, distinctParent method receives *XMLObj* with LINQ to retrieve nodes without repeating

Algorithm: *XML Optimizer*

Input Data : origin XML file path

Result: new XML file structure without repeated tags

```
1  begin
2      call XmlUniqueSructure( )
3       define file ←IO namespace
4       define  fileData ← new string /* StringBuilder recommended */
5      fileData.Append("<").Append(root).Append(">")
6      for i=0 to Parent[].count Loop
7        fileData.Append("<").Append(Parent[i]).Append(">")
8        for each child in parent.Childs[] Loop
9          fileData.Append("<").Append(Child[i].Name).Append(">")
10         fileData.Append(Child[i].data)
11         fileData.Append("</").Append(Child[i].Name).Append(">")
12       end loop
13     end loop
14      fileData.Append("</").Append(root).Append(">")
15      file.AppendAllText(fileInputPath.xml, fileData)
16    end if
17 end
```

This algorithm is used to build the new optimized XML file structure based on *XmlUniqueSructure* algorithm. StringBuilder is proposed mechanism to build strings based on its high speed of building strings [8]. In this algorithm the origin xml file nodes are traveled depend on collected parents and Childs using dynamic *XPath* which is created by *XmlUniqueSructure* algorithm to ensure rapidly speed access to each node. Finally, the new optimized xml file is saved and become ready to use.

Experiments and result

The experiments have been done to test the impact of XMLOptimizer on RT, CPU utilization and memory space factors.

1- Response Time (RT)

Response Time: is the vital computing factor in any system/application [15]. RT According to the IBM Dictionary of Computing is “The elapsed time between the end of an inquiry or demand on a computer system and the beginning of a response" [6]. RT is used to measure time from submit request until get first response [16].

The experiments have been done about 9 times on XML files. XML files contain thousands of records as in Figure (4).The size of XML files that are measured starts from 293KB until 93MB.This experiment measures RT of search operation on Optimized XML files by XML Optimizer of WSMM component and others without optimization as well as

measure enhancement rate of search operation.

Experiment 1: *Measure RT of search operation from optimized XML file*

Input: XML files with different size starts with 293KB to 93MB.
Result: comparison between RT of optimized and none optimized XML
Size Unit: Second

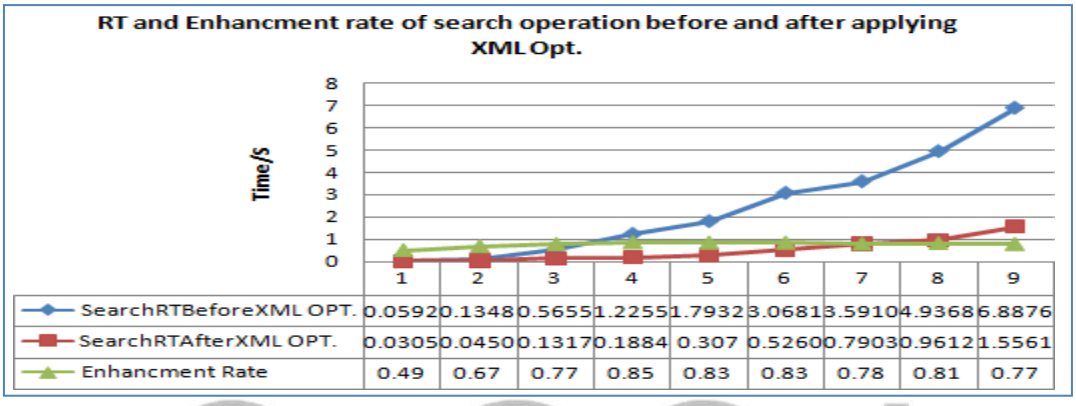


Figure (4): RT of Search Operation after and before applied XML Optimizer

Result Analysis:

The previous Figure shows that RT of search performance enhanced with the XML Optimizer. That is noted with enhancement rate

Experiment 2: *Measure RT of XML files optimizing operation*

This experiment has two parts; in part ***I*** the experiment is done on Windows7 with Intel core i3; in the other side, part ***II*** done on Intel Core i5.

Table (2): Experiment environment settings

Environment	Part I	Part II
Windows	7	7
M.P	Intel Core i3	Intel Core i5
File Size/M	0.29-93	1.54-312
Records No.	1141- 353,710	24200 - 4,800,200
Tables No.	Single table	Multiple Tables (3)
Exp. Times	8	7

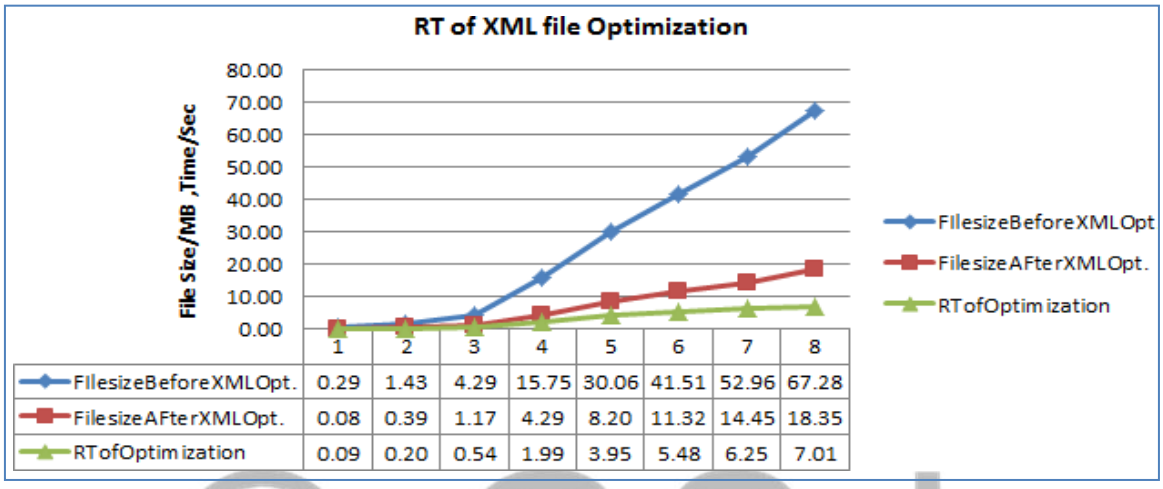
Part I:

This experiment has been done 8 times; each time is performed on different XML file Size as in table (2).

Input: 8 XML files with sizes between .29 MB and 93MB.

Result: RT of compression operation as well as compression ratio

Size Unit: MB



	1	2	3	4	5	6	7	8
FilesizeBeforeXMLOpt.	0.29	1.43	4.29	15.75	30.06	41.51	52.96	67.28
FilesizeAFterXMLOpt.	0.08	0.39	1.17	4.29	8.20	11.32	14.45	18.35
RTofOptimization	0.09	0.20	0.54	1.99	3.95	5.48	6.25	7.01

Figure (5): Comparison between xml file before and after applying XML Optimizer

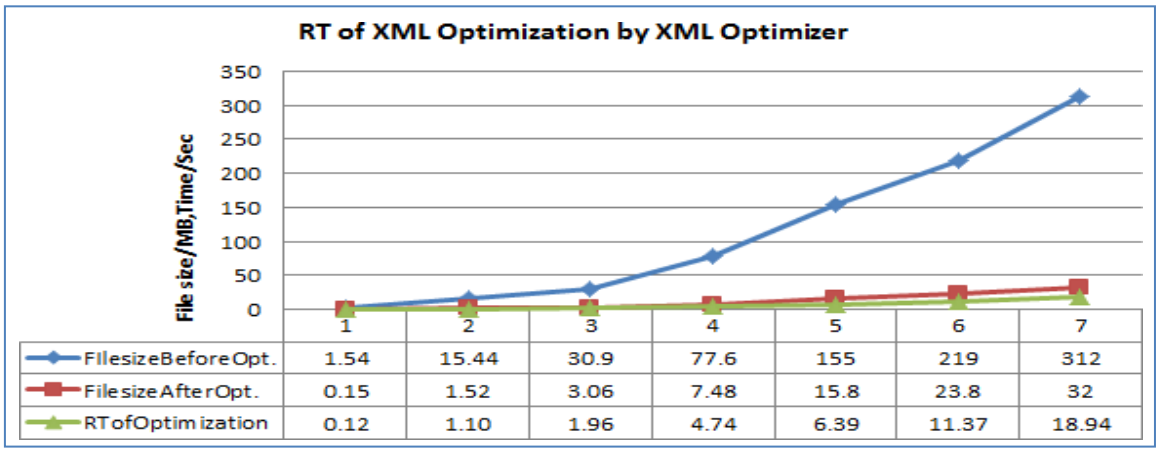
Part II:

This experiment has been done 7 times; each time is performed on different XML file size as in table (2).

Input: 7 XML files with sizes between 1.54 MB and 312 MB.

Result: RT of compression operation as well as compression ratio

Size Unit: MB



	1	2	3	4	5	6	7
FilesizeBeforeOpt.	1.54	15.44	30.9	77.6	155	219	312
FilesizeAfterOpt.	0.15	1.52	3.06	7.48	15.8	23.8	32
RTofOptimization	0.12	1.10	1.96	4.74	6.39	11.37	18.94

Figure (6): Comparison between xml file before and after applying XML Optimizer

Result analysis:

The results in Part I are enhanced in Part II based on experiment environment, which become Intel Core i5, although the size of XML files become larger than in Part I. As demonstrated, the modern environment, the more impressive results.

2- CPU Utilization

CPU Utilization indicates extend of CPU utilization to perform the processes [17]. Thus, CPU Utilization refers to CPU Utilization [18].Practically; efficiency of the proposed model implies high CPU Utilization.

Experiment 1: *Optimized vs. Non-Optimized CPU Utilization of search operation*

Another test has been done in Figure (7) bellow and shows the CPU Utilization, when search operations are achieved on XML files before and after are optimized. This experiment has been done 8 times on 8 different XML files size.

Input: 8 XML files are differenced in size before and before XML optimizer is applied **Result**: comparison between RT of search operation on XML files before and after reduces file size

Time Unit: Second

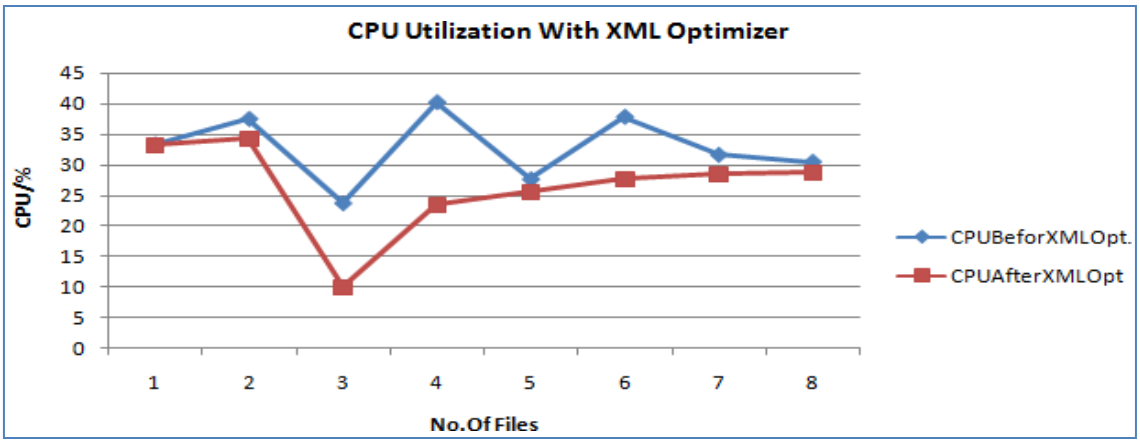


Figure (7): CPU Utilization on optimized and Non-optimized XML files.

Result Analysis:

Previous Figure shows the enhancement of CPU Utilization, when XML Optimizer is applied.

3- Memory Space

Memory Space indicates the amount of memory is needed to store either LCC or temporary notifications files. Based on [2] Memory is "central to the operation of a modern computer system. Main memory is a large array of bytes, ranging in size from hundreds of thousands to billions".

Experiment 1: *Measure XML file size with XML Optimizer*

This experiment has been done on many XML files of LCC with difference size to measure applying XML Optimizer on them. These XML files contain one table. The experiment has been done for 9 times, each time is performed on different size of XML file.

Input: 9 xml file with different sizes in KB where (269.96 KB≅1143 record as in Figure (8), as well as there is one table in XML file
Result: comparison between XML file size before and after applying XML Optimizer
Size Unit: MB

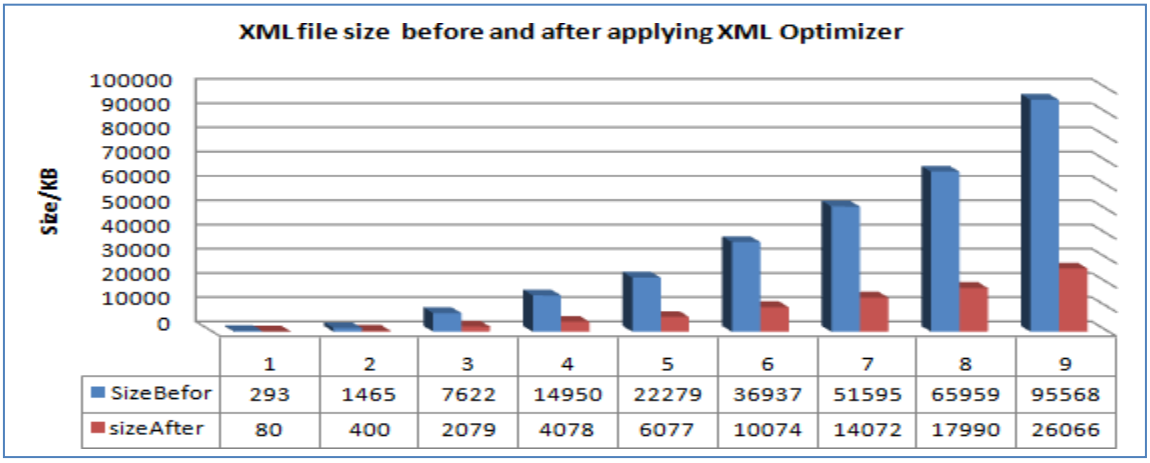


Figure (8): File size after and before apply XML Optimizer

Result Analysis:

The previous figure shows the difference between XML file before and after applying XML Optimizer on them, where XML file is reduced to73%.As a result, the XML optimizer saved the memory space efficiently.

Experiment 2: *Optimize XML file size with Multiple Tables*

This experiment has been conducted on many XML files with multiple Tables (Parents) to measure extend of impact XML optimizer on these types of files. The experiment is done on small, medium and large files as in table (3).Table (3) below shows chosen sample of three XML files contain data.

Table (3): DB Tables and count of records for each table in XML files

Tables/Records	Small-1.54 MB	Medium-77.6 MB	Large-312 MB
Students/rec	600	300,000	1,200,000
Subjects/rec	200	200	200
Results/rec	1800	900,000	3,600,000

The experiment has been done for 7 times, in each time performed on different XML file size. The first experiment starts with 1.54 MB and the last one with 312 MB.

Input: 7 xml file with different sizes in MB contain three tables with thousands records
Result: comparison between XML file size before and after applying XML Optimizer
Size Unit: MB
Environment: Windows 7, Intel Core i5

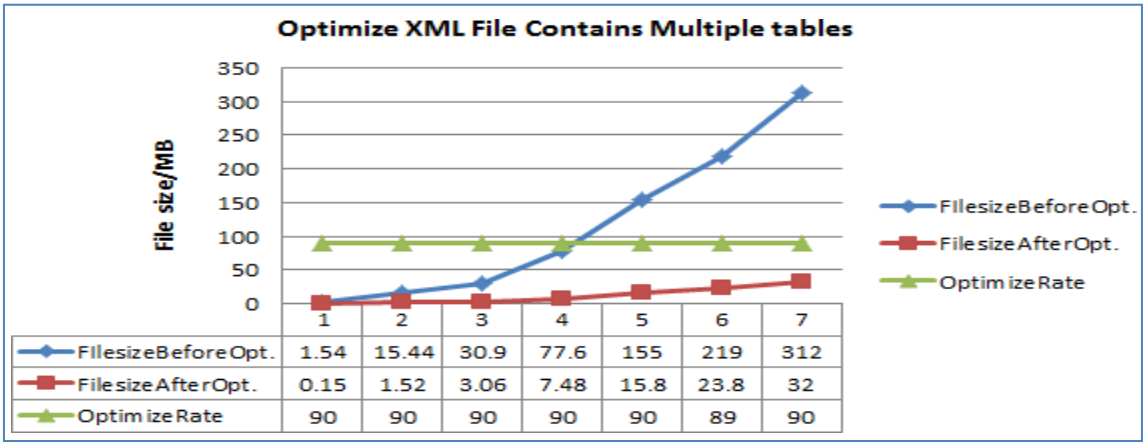


Figure (9): XML Optimizer with Multi-tables

Result Analysis:

The Figure shows the rate of optimization of xml files after apply XML Optimizer. This rate reached to 90%. That means the XML Optimizer performance increased with XML files that contain multi-tables. In the general, Optimization rate depend on some vital factors such as: length of tag name, size of inner data within tags, count of distinct tags.

Compare with other studies results

In this section of paper, there is mention about comparing WS model with XMLOptimizer and previous study XPRESS as in Table (4).The table contains summery of comparison that illustrates extends of enhancement with XMLoptimizer.

Table (4): compare XPRESS and XMLOptimizer performance

Study	Comparison	Rate	XML content	Optimizing method
XPRESS	XML Compress	73%	Single table	Compress/decompress
XMLOptimizer	XML Compress	76%	Single table	Restructure
XMLOptimizer	XML Compress	90%	Multi-tables	Restructure

This comparison based on experiment has been done on both traditional WS model and WS model with XMLOptimizer. The results in figure (10) illustrate the enhancement in WS with XMLoptimizer over traditional WS model.

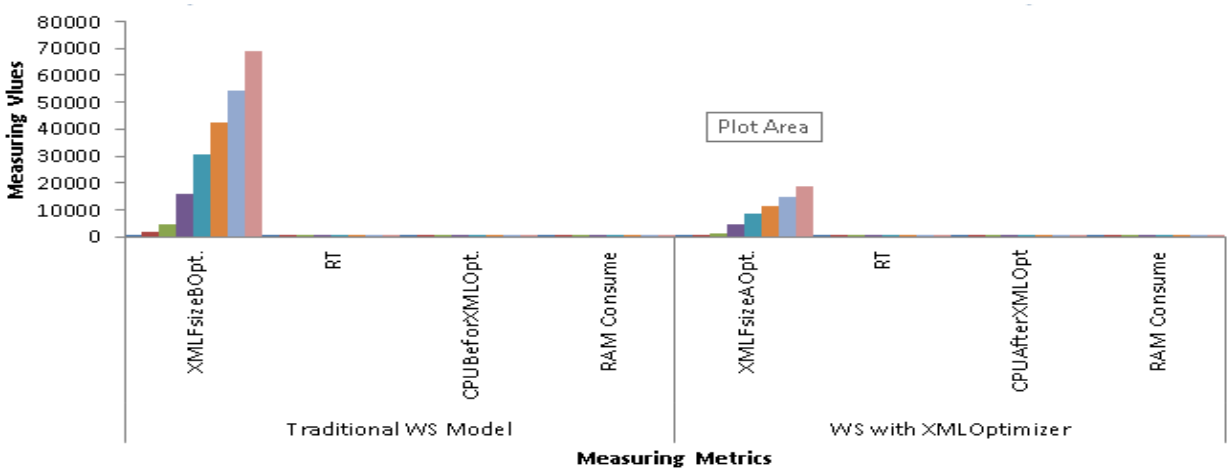


Figure (10): compare WS with XMLOptimizer performance with Traditional WS Model

Conclusion

In this paper, XMLOptimizer was underwent several experiments to evaluate its performance. Based on the analysis of the results, XMLOptimizer is noted to save significant system resources by re ducing RT, CPU usage and memory space.

The size of XML files has been reduced by 90%, which is better than other methods for compressing XML files. In addition, RT of operations such as search and insert that have been achieved on XML files enhanced about 99%.So XMLOptimizer proved to be effective in improving the SOAP WS model.

References

1. Bean, J., *SOA and Web Services Interface Design*. 2010: Elsevier Inc.
2. Roy Grønmo, D.S., Ida Solheim, Jon Oldevik, *Model-driven Web Services Development.* IEEE, 2004.
3. Connolly, T., C. Begg, and R. Holowczak, *BUSINESS DATABASE SYSTEMS*. 2008: Pearson Education Limited.
4. Abilio, C.M.G.a.R., *Systems Integration Using Web Services, REST and SOAP: A Practical Report.*FSMA, 2017. 19: p. 34-41.
5. Dustdar, S. and W. Schreiner, *A survey on web services composition.* Int. J. Web and Grid Services, 2005. 1(1).
6. IBM. *Response Time*. 2017 [cited 2018; Available from: 6http://searchnetworking.techtarget.com/definition/response-time
7. A. Bhuvaneswari, G.R.K., *Semantic web service discovery for mobile web services.* Int. J. Business Intelligence and Data Mining, 2018. .
8. Rohit Ranchal, B.B., Pelin Angin, Lotfi ben Othmane, *EPICS-A Framework for Enforcing Security Policies in Composite Web Services.* IEEE, 2018.
9. JunKi, et al., *XPRESS: A Queriable Compression for XML Data.* 2003.
10. Liefke, H. and D. Suciu, *XMill: an Efficient Compressor for XML Data.* ACM.
11. TSALGATIDOU, A. and T. PILIOURA, *An Overview of Standards and Related Technology in Web Services.* Kluwer Academic, 2002. 12.
12. Mumbaikar, S. and P. Padiya, *Web Services Based On SOAP and REST Principles.* International Journal of Scientific and Research Publications, 2013. 3(5).
13. Bedi, K.S. *REST Vs SOAP*. 2013 [cited 2017; Available from: http://www.c-sharpcorner.com/blogs/rest-vs-soap1
14. Wagh, K. and D.R. Thool, *REST and Soap Compare.* Journal of Information Engineering and Applications, 2012. 2.
15. Inc, T. *Response Time*. 2017 [cited 2017; Available from: https://www.techopedia.com/definition/9181/response-time.
16. SILBERSCHATZ, A., P.B. GALVIN, and G. GAGNE, *OPERATING SYSTEM CONCEPTS*. 9 ed. 2013: wiley.
17. Inc., T. *CPU Utilization*. 2018 [cited 2018; Available from: https://www.techopedia.com/definition/28291/cpu-utilization
18. Microsoft. *Interpreting CPU Utilization for Performance Analysis*. 2018 [cited2018;Availablefrom: ttps://blogs.technet.microsoft.com/winserverperformance/2009/08/06/ interpreting- cpu-utilization-for-performance-analysis/.