The Effects of Data Compression on Performance of Service-Oriented Architecture (SOA)

Hosein Shirazee¹, Hassan Rashidi² and Hajar Homayouni³

¹Department of Computer, Qazvin Branch, Islamic Azad University, Qazvin, Iran
²Department of Computer, Qazvin Branch, Islamic Azad University, Qazvin, Iran
³Department of Computer, Al-Zahra University, Tehran, Iran

Abstract: Due to the various areas of applicability, there is a need for overcoming performance problem of service oriented architecture. In this paper, the impressive parameters on response time in service oriented architecture are determined. One of the important factors in reducing response time is the size of data being transferred in the network. Therefore, in this research the effect of using compressing algorithms on response time is analyzed. The experimental results obtained using two compressing algorithms and XML dataset show that using compressing algorithms has a deep impact on reducing response time and increasing the whole performance of service oriented architecture.

Keywords: Service Oriented Architecture Performance, Compressing Algorithm, Response time

1. INTRODUCTION

Service oriented architecture as a novel and complete way of implementing distributed architecture through services with loose coupling is applicable for different organizations and applications. The main problem of this architecture is its performance which is the result of being loose coupling and its heterogeneous nature. Due to the various areas of applicability, there is a need for overcoming performance problem of service oriented architecture. For this purpose, the reasons of this problem should be analyzed and some well-defined metrics for performance measurement should be indicated [1].

In this paper, the effective parameters on response time in service oriented architecture are indicated first, and shows that one of the important factors in response time is the size of data being transferred. Therefore, the effect of using compressing algorithms on response time is analyzed. The experimental results obtained using two compressing algorithms and XML (as a data format for web service infrastructure) dataset show that using compressing algorithms has a deep impact on reducing response time and increasing the whole performance of service oriented architecture. The rest of the paper is organized as follows. Section 2 speaks about some prerequisites for the research. Section 3 is our proposed method and section 4 and 5 are obtained results analysis. Finally section 6 is about future works.

2. PREREQUISITES

2.1 XML in service oriented architecture

XML is a data format for web service infrastructure. It is an open standard and flexible format which can represent various types of text data. XML documents are readable by human beings which is much better than binary formats. Furthermore, metadata can be embedded between XML documents. However, using XML as data representation format in web services leads to data processing and transferring overhead.

The size of XML messages is 10 to 20 times larger than binary form of data, thus their transferring time on networks is too long. As XML is based on text, a preprocessing on it is needed before any kind of operations which includes at least 3 following activities that all need CPU and memory:

- Decomposition: transforming XML data to correct structure of components used XML. This decomposition includes lots of processes on strings.
- Validation: a step before or through decomposition phase for making sure about the correct structure of received data. This phase might take even more time than decomposition phase, especially when DDT or schema is specified remotely.
- Transformation: this is transforming one XML structure to other. This phase is commonly when integration between services and components obtained from different providers is needed. This phase can reduce XML decomposition speed 10 times less and should be taken in to consideration as the first factor for increasing performance of web services[2].

Different techniques for minimizing transformation and processing cost are proposed that some of them are mentioned as follows.

- Utilization of compressing approach on XML documents: in this method both compression and decompression times should be considered.
- Using dedicated decomposition model: this approach helps to use a specified model for each decomposition ways. For example when there is a need to edit or read
a part of document, it’s better to use an document object model but if a serial reading of data is needed, it’s better to use programmatically interface for XML or there is a need to read only a part of a whole document, it’s better to read it locally instead of reading all of a document.

- Not using validation: in most of cases that there is an assurance that XML document are well-defined, there is no need to validate documents. In addition, when a XML document is validated, it’s possible to convert it to a non-included DDT or schema document. Although caching non-local DDT or schema is another way.

2.2 XML document compression

The use of compressing methods can lead to a reduction in the information transferred size and response time and therefore an increasing in the whole performance of system. Choosing an appropriate method to compress data is an important task. Generally speaking there are three different classes of XML data compression methods that are explained as follows.

1. General purpose Compression algorithm:

   Gzip [3], [4] is based on Huffman coding [5] and LZ77 algorithm [6]. Bzip2 [7] is an implementation of the Burrows-Wheeler block sorting algorithm [9]. This algorithm has three-stage process. The first stage, called block sort, is a pre-processor that makes data more compressible which is achieved by dividing the data into N blocks, and by performing all possible cyclic shifts of these blocks to form an N × N matrix. In this matrix, the rows are the N unique rotations which are then sorted in lexicographical order. The shift and sort operations bring commonly repeated strings close to each other. The row number of the original row in the sorted matrix is included; therefore the pre-processor increases the size of the data. In the second stage of the algorithm, this data is typically passed to a simple move-to-front encoder, where each symbol that was previously seen is encoded using the distance from where it was last seen. The last stage of the algorithm is a statistical compressor (e.g., Huffman coding). The LZ77 algorithm is a substitution-based compressor. In this algorithm, data sequences previously seen are replaced with (1) a tuple including a reference to the previously seen data and (2) the length of the sequence. In LZ77, the previously seen data is limited to data available in a fixed-size read buffer which is referred to as a window, and the reference to previously seen data is given relative to that window.

2. XML Aware Compression Algorithm:

   There are two important problems when treating XML as general-purpose data, both due to the fact that XML messages are generally more structured than most text files: (1) XML documents include two very different kinds of data: markup and payload. (2) There is a reason for using markup in XML documents, and by compressing it as if were payload, we reduce the usefulness of the document. For instance, an XML document compressed with Gzip cannot be queried using XQuery [10] without first decompressing the entire document. The simplest XML-aware compressors are substitution-based algorithms that work at the markup level. BXML [11] is an example which is an adaptive algorithm that maintains a dictionary of already seen tags, attributes and namespace prefixes while scanning an XML document. When a previously seen entity is encountered, it is replaced by a single byte. This byte is an index in the embedded dictionary. The compression algorithm compresses only the markup of the XML document. BXML is a simpler version of the Wireless Application Protocol (WAP) Binary XML (WBXML) [12]. A major difference between BXML and WBXML is that the dictionary of WBXML also includes well-known attribute values.

   The main problem with both BXML and WBXML is the limited token space that one byte can offer. BXML solves this by using limited memory and by replacing dictionary entries as needed using a FIFO strategy. WBXML uses different code spaces, each divided into code pages. Therefore, the meaning of a byte token is dependent on the code space and code page currently in use. Millau [13], [14] is for improving WBXML. A major difference between these two compression schemes is that Millau separates the XML markup from payload by putting them in different streams. Millau further tries to solve the problem of the limited token space of WBXML in two ways: (1) by optimizing the scheme used by WBXML and by trying to minimize the code page switches; (2) by trying variable byte encoding, where the frequency of a token effects on the number of bytes needed to encode it. The frequency is either guessed from the XML Schema Definition (XSD) or the Document Type Definition (DTD) file, or calculated while analyzing the XML document.

   While improving on the simple substitution-based WBXML, Millau attempts to separate markup from payload. The XMill [15] compressor uses a similar strategy and a simple binary encoding for the markup of the document. Based on tag names, the payload is then divided into different containers. It is assumed that related data items in the payload will show a greater redundancy when grouped. Each container can utilize its own compression scheme, including custom-made compressors.

   XMLPPM [16] is a Prediction by Partial Match-based (PPM) compressor algorithm [17]. This algorithm uses the input data to build a statistical model and exploits this
model to generate a probability distribution which can help predict the next symbol and is then used by an arithmetic coder [18]. XMLPPM first transforms the XML document into Encoded SAX (ESAX), which is a binary version of the Simple API for XML (SAX) similar to BXML. Then, this ESAX representation serves as input to several multiplexed PPM encoders.

All above-mentioned compression algorithms have the same problem which is the lack of support for queries. Many compression algorithms (e.g., LZ77) are adaptive. While adaptive algorithms do not require several passes to compress a file, they present the disadvantage that the representation of a compressed string depends on the actual location of the string in the file. Therefore, to support queries, one must be able to search the compressed payload. XGrind[19], which is another XML compressor, solves this problem by separating markup from payload, and then applying a non-adaptive Huffman-based compression scheme for the payload. The XCQ [20] system uses an adaptive compression scheme, but divides the payload into smaller blocks that are compressed independently of each other. This way, only relevant blocks need to be decompressed before they can be queried.

3. Schema-aware Compression Algorithm:

The previous compressors exploit the separation between XML markup and payload, but note the fact that the markup of an XML document can be further specified by a schema defined in an XSD or a DTD file. These schema-aware methods do not encode the parts of the XML Infoset that can be reconstructed by the decoder of the receiving party. These encoders and decoders are automatically generated from the XML Schema; much like a compiler front-end can be generated from a grammar. For instance, the encoder/decoder approach can help to remove tag names and namespace prefixes. Millau includes a schema-aware compression scheme known as Differential DTD, in which only the non-predictable parts of the DTD are included. Examples of non-predictable parts consist of choice, optional elements, and unbounded lists. Levene and Wood [21] also suggest a similar approach.

Schema-aware encodings consist of Fast Schema and Bin-XML. Fast Schema is part of Sun’s Fast Web Services technology [22]. An XML document representing a Web Service message is serialized into a corresponding document in Abstract Syntax Notation One (ASN.1). The ASN.1 description is generated from the XML Schema, which in turn is retrieved from the WSDL specification of a Web Service. Another schema-aware method is Bin-XML [23] which supports dynamic updates and random access.

Schema-aware methods generally present better opportunities to achieve a higher compression ratio, since they can drop information which is not necessary. However, they all rely on the schema, so if this schema changes, all the encoders and decoders must be regenerated and deployed. This can require recompilation and redeployment, depending on the structure of the software. This makes them application specific.

3. PROPOSED METHOD

In this research the effective parameters on response time of web services used in service oriented architecture and their effects are analyzed. Due to the fact that the messages transferred between web services are in standard XML format and this format is too long, it is necessary to compress this information. In other words, in this research it is claimed that one of the effective factors on response time in service oriented architecture is the length of messages to be transferred. In this paper, the influence of using compression algorithms on response time in service oriented architecture is studied in real experiments.

3.1 Effective parameters on response time

In this part parameters which have influence on response time in service oriented architecture are indicated separately and the effect of data compression in response time when it is applied on source and destination is analyzed. The response time is defined as the time needed for calculating the response by a web service.

- Encoding and decoding time:

Due to the fact that the Standard XML format is utilized for data transitions, the encoding and decoding time is composed of the time needed for data preparation in standard XML format and also data extraction on destination and vice versa.

- The number of transferred messages:

Any services can response to a limited number of requests and any requests use some hardware resources and in some cases services cannot response to different requests simultaneously. Therefore the number of transferred messages can effect on response time in service oriented architecture. For simplification, the simultaneous messages can be ignored and consider that only one request is sent to the destination each second.

- Average compression rate:

Different compressing algorithm differs in compression rate. The higher the compression rate of algorithm is, the lower the transferring data time gets. This leads to a reduction in response time and an increasing in throughout of web services and finally results in performance of service oriented architecture.
The number of processes needed for transferring in HTTP and TCP protocols:
Using HTTP and TCP protocols for transferring messages, the number of processes involved effect on message transferring time in source and destination.

Passing time from network components:
A local network contains some components such as routers and other equipment. The needed time for passing from these components are sometimes considerable.

The geographical distance:
Considering service oriented architecture as a kind of distributed architecture, the geographical distance between service provider and service invoker might be too long. Suppose that the information are transferred in light speed in long distances, thus in small distances this time will be almost equal to zero.

### 3.2 Data compression effects analysis

- Choosing an appropriate data set:
  Since there is no training data set for web services in web service community, in this research we prepared a standard data set for our tests. This data set includes some SOAP messages (for instance XML documents) which are real data and are used in other experiments. Two main ideas for creating this data set are:
  1. The data set should contain real world data
  2. The data set should be common in web services messages.

  For this purpose we use a dataset which concerns with database invoked by web services. This dataset is messages related to a web service invocation which provides the access to a database. For accessing to this database, the web service receives an SQL query as an input to read from database and run it on the database and the resulted output is returned in XML format. Each column is tagged in this output. These tags are repeated for each extracted record from database.

  The size of resulted document varies from 700 B to 130 KB. This leads to an acceptable dataset for XML messages in real world.

- Choosing appropriate compressing algorithms:
  Two different open source algorithms i.e. BXML and Bzip2 are selected for doing our experiments. There was an implementation for Bzip2 algorithm in .Net framework. Since no implementation was found for BXML, we implement it in .Net framework.

### 4. EXPERIMENTAL RESULTS

In this part the results obtained using two compressing algorithms and XML dataset are analyzed. The algorithms are implemented using .Net framework. The performance of data transfer using compression is studied. For this purpose both algorithms are executed in the same conditions using the same platforms. First of all the percentage of compression using these algorithms are calculated (table 1 and figure 1) and then the needed time for compression or decompression processes is indicated (table 1 and figure 1). Each algorithm was 15 minutes in run mode in order to obtain more accurate results. Furthermore, 5 data packs with different size but all related to web service invocation are used to increase accuracy and indicate the effect of data size on compressing algorithm.

#### Table 1: The percentage of compression using Bzip2 and BXML algorithms

<table>
<thead>
<tr>
<th>Compressing Algorithm</th>
<th>Data Size(KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.7</td>
</tr>
<tr>
<td>Bzip2</td>
<td>69</td>
</tr>
<tr>
<td>BXML</td>
<td>81</td>
</tr>
</tbody>
</table>

#### Table 2: The compression time according to data size

<table>
<thead>
<tr>
<th>Compression Algorithm</th>
<th>Data Size(KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.7</td>
</tr>
<tr>
<td>Bzip2</td>
<td>.03</td>
</tr>
<tr>
<td>BXML</td>
<td>.01</td>
</tr>
</tbody>
</table>

#### Figures:

- **Figure 1:** The percentage of compression using Bzip2 and BXML algorithms
- **Figure 2:** The average response time of web service using compressing algorithm
5. RESULTS

This research is based on this fact that using XML standards in Service Oriented Architecture and large amount of data in this standard will cause low performance in this system. Because of that, using compressing algorithm will increase response time and performance on Service Oriented Architecture.

This research has presented a new way to decrease response time in Service Oriented Architecture. This technique includes using a compression algorithm and improving response time in this architecture. With analyzing experiments results, we can approve that:

- Using compressing algorithms when data size is more than a threshold may increase response time and improve performance, but when data size is no more than a threshold, compression’s overhead causes increasing response time.
- Network bandwidth has a positive effect on using compression algorithm. In lower bandwidth network, compression ratio and decreased data size can have a less threshold. But higher bandwidth network needs a higher threshold.
- In order to compress Bzip2 and BXML and experiments results, can inference that Bzip2 algorithm has a better compression ratio. When BXML algorithm was used, compression ratio was never more than 45 percent. In the other side, when Bzip2 was used 6 percent compression ratio was achieved.
- Although BXML algorithm’s compression ratio is less than Bzip2 algorithm, it has less processing time in average. For example for compressing a specified data with a size of 100 KB through BXML algorithm 3 sec is consumed whereas for compressing same data using Bzip2 9 sec is needed.
- In experiments with data size of 65 KB, both of two different ways has an equal time. Since although compressing this size of data hasn’t an equal time, BXML has a less consumed time for compressing data in comparison with Bzip2.

6. FUTURE WORKS

Due to research experiments and related works we can suggest following ways for future:

- Using other Compressing Algorithms:
  We have considered 2 different XML Data compressing algorithms and as a future work, it’s good to use other algorithm to compare with these ones.
- Studying Encryption methods affection:
  Using encryption algorithm for transferring XML data is regular and will have effect on performance and compressing ratio. In this case, Studying encryption method and finding a good method for compressing data can be very important as a future work.

References

Management of Data (SIGMOD), 2000.


