IJCSN International Journal of Computer Science and Network, Volume 3, Issue 4, August 2014
ISSN (Online): 2277-5420 **www.IJCSN.org**
Impact Factor: 0.274

JSON vs XML: A Comparative Performance Analysis of Data Exchange Formats

[1] Saurabh Zunke, [2] Veronica D'Souza

[1] Deloitte Consulting US India Pvt Ltd
Hyderabad, Andhra Pradesh 500081, India

[2] Vishwakarma Institute of Information Technology, University of Pune
Pune, Maharashtra 411048, India

Abstract – Service Oriented Architecture is integrated in the very fabric of the nature of Internet. The Internet as it exists today is made up of numerous components which operate asynchronously. These independent components communicate with each other using a specific set of formats that standardize the communication and regulate the coherence of the communication. The SOA structure is made up of multiple loosely coupled components that can be broadly classified as service producers and service consumers. The loose coupling allows enterprises to respond to changes quickly by updating, replacing and changing concerned modules with flexibility, without affecting other components coupled to it. This paper first compares the features of the two most widely used Data Exchange formats for communication between these components and finally, provides a comparative analysis on the performance of these formats using benchmarks.

Keywords – *Service Oriented Architecture, Comparative Performance Analysis, Data Exchange formats.*

1. Introduction

This section will briefly explain Data Exchange formats, the more widely used XML and the comparatively new JSON, their design goals and their fundamental characteristics.

1.1 XML

XML or the Extensible Markup Language is a set of standards published by the XML Working Group under the World Wide Web Consortium (W3C) – the group that regulates the standards of the Internet (www). Basically, an XML document consists of some well-defined markups called tags, and the data enclosed within them. The tags are containers which describe the enclosed data and also organize it. The design goals for XML v1.1 as published in the W3C recommendation [1] are as follows:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

The syntax specification and regulations regarding the well-formedness of an XML document can be looked up in the W3C specification for XML v1.1 [1]

```
<?xml version="1.1"?>
<greeting>Hello,world!</greeting>
<!--This is a comment -->
```

Fig. 1 Sample XML document

1.2 JSON

JSON or JavaScript Object Notation [3], is a light-weight text-based open standard designed for human-readable data. It is the most widely used format for interchanging data on the web after XML. It originates from the JavaScript language and is represented using two primary data structures: ordered lists (recognized as 'arrays') and name/value pairs (recognized as 'objects'). The JSON standard is language-independent and its data structures,

arrays and objects, are universally recognized. These structures are supported in some way by nearly all modern programming languages and are familiar to nearly all programmers.

JSON has been developed to make it as similar as possible to the data structures used in day to day computing and hence it is easier for the computer to operate on it. But this same benefit deems it less human-readable and less flexible as compared to XML.

The syntax specification and regulations regarding the constructs of a JSON object can be looked up in the Internet Engineering Task Force specification for JSON notations [2]

```
{ "greeting": "Hello,world!" }
```

Figure 2: Sample JSON object

Note: There is no provision for writing comments in JSON.

2. Feature Comparison

This section will talk about the salient features of both these data exchange formats and compare them vis-a-vis each other.

2.1 Industry Adoption

- XML has been the industry standard for around a decade and has a lot of supporting frameworks and standards to govern the implementation.
- JSON on the other hand is relatively new on the block and does not have standards like Schematrons, XSDs, to govern its implementations.

2.2 Human Readability

- XML documents are easy to read for humans.
- JSON is highly cryptic due to the use of parenthesis delimiters.

2.3 Metadata

- XML has a big overhead in the form of tag metadata.
- JSON has minimal metadata making it compact but a loosely defined format.

2.4 Supporting Frameworks

- XML is supported by a majority of framework implementations (API) across the industry.
- JSON lags behind but is catching up very quickly in terms of support adoption.

2.5 Extensibility

- XML allows you to store any data type. Data element attributes allow additional flexibility.
- JSON is limited to only storage of classical data like text and numbers.

2.6 Ease of Mapping

- XML is document oriented (Tree Structure) making it difficult to map it to objects in the OOP paradigm (Graph structure).
- JSON is data oriented [8] and is hence closer to the graph structure of objects in OOP paradigm.

2.7 Bandwidth Performance

- Due to metadata overhead, the same data takes more bandwidth if expressed in XML.
- JSON data is highly compact using least amount of bandwidth.

3. Benchmark Comparison

This section will talk about the performance comparison between both these data exchange formats and will conclude with the scenarios each of them is better suited for.

We will look out for the following parameters that play a major role in determining the memory and bandwidth performance of a SOA application:

- Marshalling time: This is the time taken, in nanoseconds, by the JVM to convert an object into a stream of data
- Unmarshalling time: This is the time taken, in nanoseconds, by the JVM to construct an object from a given stream of data.
- Stream size: The size of a marshalled object in Bytes.
- Memory footprint: The total memory in Bytes used during runtime from the JVM heap.

3.1 Performance Analysis

For performance analysis and comparison, both data formats were benchmarked against memory usage and runtimes in the following three experiments:

1. Marshalling from a Java object to a buffered memory stream without compression

2. Marshalling from Java object to a buffered memory stream with compression
3. Unmarshalling from a buffered memory stream to Java object.

The tool used to achieve this was Caliper [4], an open source framework made and maintained by Google for making Java Micro benchmarks. The source code and documentation for the tool can be accessed on Google's website. The marshalling tool used for XML was JAXB API [6]. JAXB (JSR 222) [10] is the standard marshal-unmarshalling toolkit recommended by Oracle and distributed with the standard version of the JDK distribution.The Jackson API for JSON [5] which is a high performance [9] JSON processor and a de-facto standard used across the industry.

The project setup was a simple Maven exoskeleton project and the data used for performing these experiments was a Java object with fixed length strings and fixed digit numerical data types which were randomly generated at runtime. The project setup had minimal code and was set up for having a deterministic memory allocation [4] and measured work pattern for every run with almost zero variance. The compression tool used was GZip [7], the standard open source zip library that is bundled with any Java distribution package.

The benchmark ran 100 sample sets for each experiment and 100 instances of every experiment. Measurements were made across each sample set aggregating them into the final result for one instance of an experiment. These results were subsequently aggregated to get the final result for a set up. This ensures normalization of underlying factors such as OS latency, processor contention and any undulation the JVM may encounter while running the benchmarks.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Hotel xmlns="http://www.example.org/Hotel">
  <hotelID>15013</hotelID>
  <hotelName>rhhjwhuqabhpitcewkrc</hotelName>
  <menu>
    <food>
      <name>wdppncqtkhubtvkagpvm</name>
      <price>48161</price>
      <calories>49311</calories>
    </food>
  </menu>
</Hotel>
```

Figure 3: Sample randomly generated XML file

```
{"hotelID":80814,
"hotelName":"glovcnbwaviboawddgvx",
"menu":
{"food":[{"name":"fprgunyzwvjkruwalbny"
        ,"price":63463,"calories":95471}]}}
```

Figure 4: Sample randomly generated JSON file

3.2 The First Benchmark – Marshalling Without Zip

The first run of the benchmark made was for marshalling of Java objects into streams (uncompressed) of formatted data which were buffered for performance optimization. The results of the benchmark run can be accessed here: https://microbenchmarks.appspot.com/runs/1c2ab2c3-7aca-4df6-b9ab-899064e28212

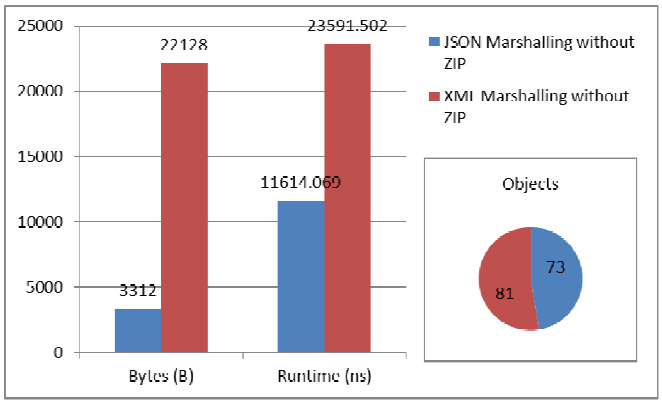

Figure 5: First Benchmark Results

The results show that there is an evident performance difference between the marshalling times favoring the JSON format, both in terms of memory usage as well as the runtime. The following observations were made during the run of the experiment:

- Stream size for JSON = 137 bytes
- Stream size for XML = 279 bytes
- Stream size ratio JSON/XML = **0.491**
- Memory footprint of JSON = **15%** of the memory footprint of XML
- Marshalling runtime of JSON = **49%** runtime of XML

3.3 The Second Benchmark – Marshalling With Zip

The second run of the benchmark made was for marshalling of Java objects into streams (compressed) of formatted data which were buffered for performance optimization. The results of the benchmark run can be accessed here:

https://microbenchmarks.appspot.com/runs/11ef2cc8-d8cc-4642-bd15-9ab58d865c35

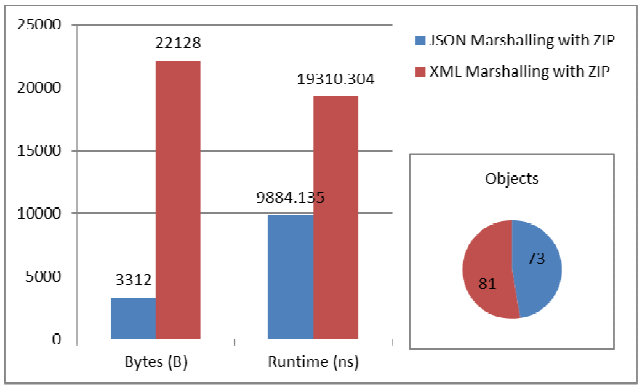

Figure 6: Second Benchmark Results

The results show that there is an evident performance difference between the marshalling times favoring the JSON format, both in terms of memory usage as well as the runtime. The following observations were made during the run of the experiment:

- Compressed stream size for JSON = 133 bytes
- Compressed stream size for XML = 217 bytes
- Stream size ratio JSON/XML = **0.612**
- Memory footprint of JSON = **15%** of the memory footprint of XML
- Marshalling runtime of JSON = **51%** runtime of XML

3.4 The Final Benchmark – Unmarshalling

The third run of the benchmark made was for unmarshalling a buffered stream of formatted data into Java objects. The results of the benchmark run can be accessed here:
https://microbenchmarks.appspot.com/runs/c97e9e09-4878-4559-973d-979f137cf158

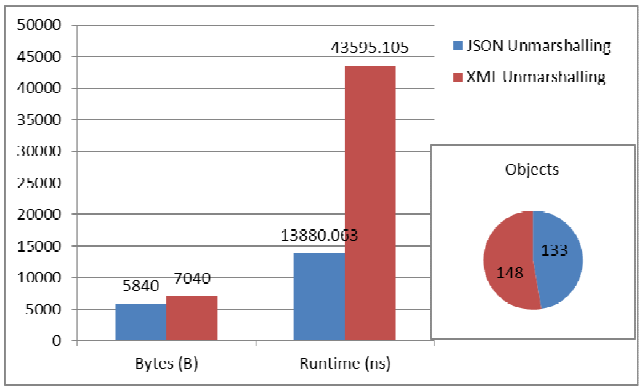

Figure 7: Final Benchmark Results

The results show that there is an evident performance difference between the unmarshalling times favoring the JSON format, both in terms of memory usage as well as the runtime. The following observations were made during the run of the experiment:

- Memory footprint of JSON = **83%** of the memory footprint of XML
- Unmarshalling runtime of JSON = **32%** runtime of XML

4. Conclusion

As far as performance is concerned, the JSON data exchange format is the clear winner between the two. In terms of both memory footprint as well as the parsing runtime, JSON delivers better performance at the cost of readability and flexibility. Even with compression enabled, XML produced a lot more overhead in the data stream than JSON. JSON lacks the attributes that help define an exchange contract between consumers and producers. This lack of contract adherence fails to maintain coherence in the communication between the two parties that exchange data.

Following table sums up the normalized conclusive findings of our experiment:

Table 1: Conclusive findings

Performance aspect	***JSON***	***XML***	***JSON/XML***
Stream size (unzipped)	137B	279B	**49%**
Stream size (zipped)	133B	219B	**61%**
Memory footprint marshalling	3312B	22128B	**15%**
Memory footprint unmarshalling	5840B	7040B	**83%**
Marshalling time (unzipped)	11614ns	23591ns	**49%**
Marshalling time (zipped)	9884ns	19310ns	**51%**
Unmarshalling time	13880ns	43595ns	**32%**

XML can be used as a data exchange format in the following scenarios:

- Performance of the application is not the primary concern
- Memory footprint of an application can be allowed to be large
- There needs to be strict adherence to a standard protocol for communication

- The data sent over the network is not of primary concern

On the contrary JSON can be used as a data exchange format in the following scenarios:

- Performance of the application is a design consideration
- Memory footprint of an application needs to be kept minimal
- There can be deviation over the format of communication between parties OR the parties agree to make necessary changes to the format as and when necessary
- The data sent over the network has to be optimized

References

[1] "Extensible Markup Language (XML) 1.1 (Second Edition)", http://www.w3.org/

[2] "The 'application/json' Media Type for JavaScript Object Notation (JSON)", RFC4627, http://www.ietf.org/

[3] "The JSON Data Interchange Format", ECMA 404, http://www.ecma-international.org/

[4] "Caliper Design Document", https://code.google.com/p/caliper/

[5] Jackson Documentation, http://wiki.fasterxml.com/JacksonDocumentation

[6] https://jaxb.java.net/

[7] "GZIP file format specification version 4.3", RFC1952, http://www.gzip.org/

[8] Zhang Yu, "Research of Conversion Method of Entity Object and JSON Data", The 2nd International Conference on Computer Application and System Modeling 2012.

[9] Ricardo Queirós, "JSON on Mobile: is there an Efficient Parser?" Dagstuhl Research Online Publication Server, Volume 38 2014.

[10] Danut-Octavian SIMION, "Java facilities in processing XML files - JAXB and generating PDF reports", Informatica Economica, Volume 12, Issue 3 2008.