Systems Integration
Marco Lucas, Bruno Almeida
University of Coimbra, DEI
Emails:
marcolucas@student.dei.uc.pt
brunoalmeida@student.dei.uc.pt

0. ABSTRACT

Serialization and deserialization (marshalling/unmarshalling) are critical processes in microservices architectures, allowing data to be transformed into a transmittable format and reconstructed at the destination. This work is ancompares these two formats in terms of processing speed (CPU time) and file size efficiency (disk space), using Java libraries JAXB for XML and Jackson for JSON.
Through controlled experiments involving the serialization and deserialization of a Java object ("Bookstore") containing varying numbers of nested "Book" elements, key performance metrics were collected and analyzed. Tests were performed with datasets ranging from 1 to 10000 elements and 100 repetitions on a MacOS Apple M1 Pro system.
The results show that JSON consistently outperforms XML in terms of marshalling and unmarshalling time, as well as generating smaller file sizes. These findings confirm the hypothesis that JSON is both faster and more space-efficient than XML for the tested scenarios, making it the preferred choice for systems where performance and storage are critical. This study provides valuable insights for developers choosing between serialization formats in microservices environments.

I. INTRODUCTION

This is a report made by the students Marco Lucas and Bruno Almeida for an assignment of the course Integration Systems. The following paper reveals the investigation that the students undertook in comparing distinct types of file formats in the given scenario: “different microservices repeatedly serializing, transmitting, and deserializing data over a Local Area Network.”

In this sense, we explore and analyze the performance of JSON and XML, two popular data serialization formats, focusing on their application in marshalling and unmarshalling Java objects. In today's software development landscape, microservices architectures are increasingly prevalent, relying heavily on efficient data exchange between services. Serialization plays a crucial role in this context, enabling the transformation of data into byte streams that can be sent over a network and subsequently reconstructed at the destination.

Knowing that, this study aims to answer key questions surrounding the performance and efficiency of those two formats.

II. CONCEPTS

To provide the appropriate context for this assignment, this section elaborates on real-life concepts, their applications, and implications.

Microservices

microservices are *"an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs;* this sharing of data between different microservices allows for providing an online service. One of the biggest real-life examples now is AWS (Amazon Web Services).

Serialization and Deserialization

Serialization or marshalling is the process of converting a data object into a format that can be easily transmitted over a network or stored. Deserialization is the reverse process, where the serialized data is reconstructed into its original form.

JSON (JavaScript Object Notation)

JSON is a lightweight, text-based data interchange format that is easy for humans to read and write, and for machines to parse and generate. It is commonly used in web applications for data exchange, particularly in RESTful APIs.

XML (Extensible Markup Language)

XML is a more verbose data format that is widely used for data serialization, especially in enterprise-level systems. While XML is more feature-rich (e.g., supporting schemas, namespaces, and validation), its complexity often results in larger file sizes and slower processing times compared to JSON. Nevertheless, XML remains relevant in many scenarios that require strict data validation or complex data structures.

JAXB

In this project, we just use one Java libraries to handle the serialization and deserialization of data:

- **JAXB (Java Architecture for XML Binding)**: A framework that simplifies the conversion between Java objects and XML. It is commonly used for handling XML in enterprise applications.

III. PROBLEM STATEMENT

For the assignment, the students formulated the following research questions:

1- “Which of the 2 formats, JSON and XML, has the best time performance in serializing and un serializing data in Java?”
2- “Which of the formats leads to the lightest file size?”

For every question, the following hypotheses were set for testing and validation:
1- “JSON is more time-efficient in marshalling/unmarshalling Java objects than XML”
2- “JSON files are lighter in disk than XML files”

IV. EXPERIMENTAL SETUP

The scenario created is described as follows: a process creates a Java object, performs marshalling and unmarshalling on it with JSON and XML, repeats the previous steps in a set number of repetitions while in each loop it measures the time.
The experiment was performed in a MacOS Apple M1 Pro 64 bit with JVM version 21.0.2. Package dependencies were handled with Apache Maven 3.9.9, while using the IDE IntelliJ IDEA CE. The Java frameworks used were JAXB and Jackson.

While the setup was being built, the initial approach was to use only the JAXB framework to compare the performance in the two formats; further research showed that JAXB’s methods, while possible to be set for JSON, were much more optimized and suitable for XML. In the end, it was decided to switch to Jackson, which is inherently built for JSON, and it would lead to more reliable conclusions about the performance differences of the 2 formats.

The following classes were implemented in the project: “Bookstore", "Book” and “Author”.
“Bookstore” contains an Array List of Book objects and it serves as the root element of the generated XML and JSON files.
“Book” has characteristics related to books and it has two versions: the first version contains an identifier (String attribute), a title (String element) and an author (String element);
the second version contains an identifier (String attribute), the number of pages (Int attribute), the date of publication (Date attribute), a title (String element) and a “Author” class object; “Author” has an identifier (String attribute), a name (String element), age (Int element) and date of birth (Date element).

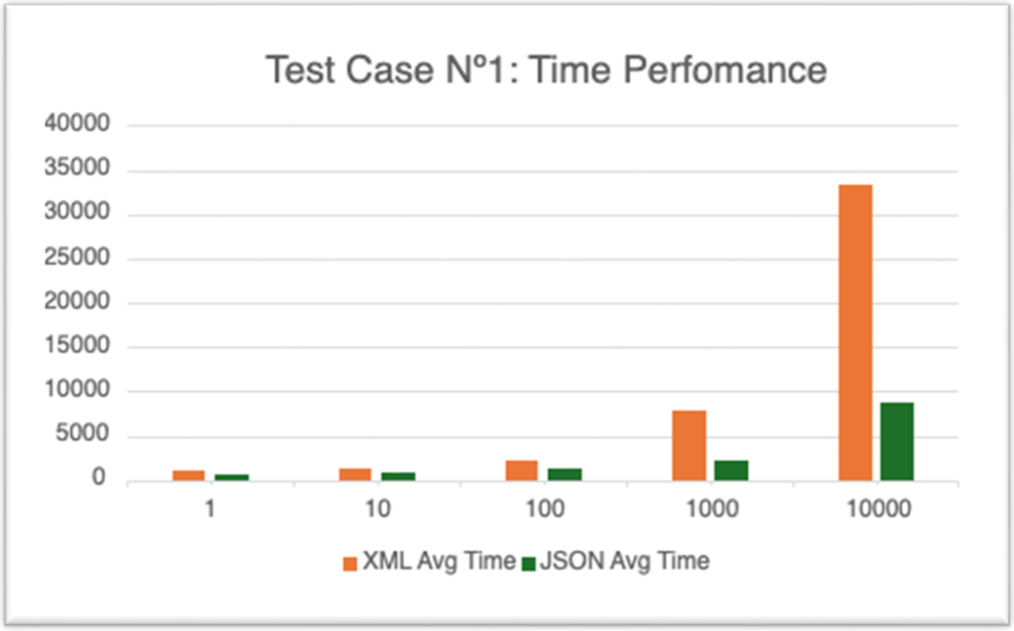
The following configurations were set for the scenario: the number of repetitions of the test case; and the number of “Book” objects inside the “Bookstore” object. For the sake of reducing the number of variables, the repetitions were set fixed to 100.

Test Case Nº1: Marshalling and unmarshalling a “Bookstore” with regular “Book” objects in JSON and XML
Test Case Nº2: Marshalling and unmarshalling a “Bookstore” with “Book” objects that contain “Author” objects in JSON and XML

V. RESULTS

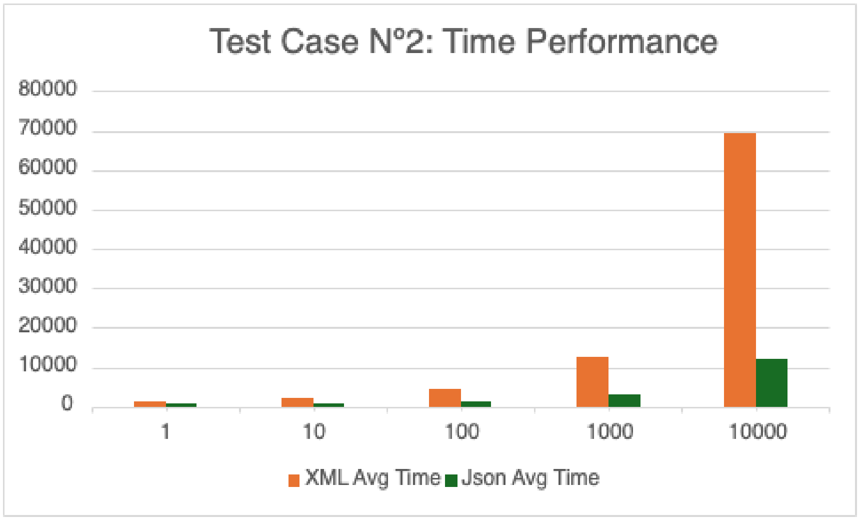
The following table shows the results of the first test case: 100 repetitions with {1,10,100,1000,10000} “Book” elements

Number of Books	XML Avg Time(microseconds)	Json Avg Time(microseconds)	XML File size	JSON File size
1	1020	680	283 B	134 B
10	1320	770	2 KB	1 KB
100	2160	1190	17 KB	12 KB
1000	7850	2080	169 KB	125 KB
10000	33370	8840	1.7 MB	1.3 MB



The following table shows the results of the second test case: 100 repetitions with {1,10,100,1000,10000} “Book” with “Author” elements

Number of Books	XML Avg Time (microseconds)	Json Avg Time (microseconds)	XML File Size	JSON File Size
1	1180	790	401 B	174 B
10	1910	940	3 KB	2 KB
100	4470	1310	29 KB	17KB
1000	12640	3030	361 KB	205 KB
10000	69350	12240	3.6 MB	2.1 MB



VI. DISCUSSION

The results show an overarching theme of JSON's performance exceeding XMLs in all the different scenarios: it becomes more noticeable when the number of elements is 10000. When the number of elements is 10000 in the 2nd Test Case, where the "Book" class has extra elements and an "Author" class, XML's time performance is, compared to the result in the first test case, an increase in more than a 100%, whereas in JSON's case, the increase is less than 50%. The analysis of the content inside the JSON files shows that the data is structured to be more compact and simplified than XML to be easily readable by humans; this improves the performance and efficiency of marshalling and unmarshalling Java objects, which corroborates with the test cases' results.

"JSON is more time-efficient in marshalling/unmarshalling Java objects than XML"

The file sizes show that JSON files are in every case lighter in disk than XML: this is easily explained by the fact that XML applies additional tags and attributes to define and describe the elements, as well as including start and end tags to signal the section of the element's definition: this naturally results in extra space in disk, while JSON stores data in key-value pairs and arrays to represent objects and values.
Therefore, the hypothesis "JSON files are lighter in disk than XML files" has been proven to be true.

VII. CONCLUSION

It can be concluded that JSON consistently outperforms XML in both time performance and file size on disk JSON's lightweight structure, combined with its more efficient serialization and deserialization times, leads to faster data handling in various scenarios. Moreover, JSON files tend to occupy less space on disk compared to XML files, making them more suitable for modern applications where performance and storage efficiency are critical. Therefore, JSON proves to be a better option for data serialization compared to XML.

REFERENCES
https://mkyong.com/java/convert-java-objects-to-json-with-jackson/
https://mkyong.com/java/jaxb-hello-world-example/