Implementation of compressed JSON for Cloud API calls

Garima Dutt[1], Anup Singh Kushwaha[2]

[1]*M.Tech Scholar, ManavRachna College of Engineering, Faridabad ,India*

[2]*Computer Science Department, ManavRachna College of Engineering, Faridabad ,India*

***Abstract*—With the option of sharing resources and storing database over the internet at different data centers, cloud has been viewed as one of the most imperative model for cost reduction and increasing economy for various enterprises. Efficient methods for structuring and formatting the data are imperative for managing and mitigating data traffic between and within cloud environments to avoid excessive bandwidth cost and to ensure interoperability. This provides better communication which is quintessential for interoperable cloud deployments. The existing data interchange formats for structuring and serializing data have not yet been analyzed in terms of efficient performance. Thus to address this issue, it is imperative to determine an appropriate data interchange format for cloud. In this paper, we have shown the performance analysis of data interchange format to assess their performance in terms of their usability in realizing a common messaging format for communicating data in Clouds. Firstly, we have explained the characteristics of each data format for clear understanding. To analyze the performance of mediation services or data interchange formats we have performed load testing in these services. Also, we have chosen a security service or an encryption technique for these mediation services which is a basic requirement to encrypt the data. It has been found that, the Optimized or Compressed JSON esteem acquired from serialization and compressed technique demonstrates an efficient mediation services for cloud API calls when contrasted with other data interchange formats. The test results in efficient mediation services which are comprehensive and have notable impact on the rate at which data is transmitted with improved performance.**

***Index Terms*—Optimized JSON, Cloud API, Encryption,**
Data interchange formats, Serialization, Cloud computing

I. INTRODUCTION

The need of data interchange formats has raised because of speediest advancement in web and cloud computing environment. It is a format for exporting and importing spreadsheet data between different programs and platforms. When two programs needs to exchange data they need to agree a common format for the data in transit. This could be a binary format, or it could be a human readable text.The binary format could be defined by one of numerous pieces of middleware, or a public format such as Google Protocol buffers. The text format could be one of the titan of data formats : XML or JSON. Or it could be something more old – fashioned, like comma separated values (CSV). Transit is defined in terms of an extensible set of elements used to represent values. The elements correspond to semantic types common across programming languages, e.g., strings, arrays, URIs, etc. When an object is written with Transit, a language-specific Transit library maps the object's type to one of the supported semantic types. Then it encodes the value into Message Pack or JSON using the rules defined for that semantic type. Whenever possible, data is written directly to Message Pack or JSON using those protocols' built-in types. For instance, a string or an array from any language is always just represented as a string or an array in Message Pack or JSON. When a value cannot be represented directly as a built-in type in Message Pack or JSON, it must be encoded. Encoding captures the semantic type and value of the data in a form that can be represented as a built-in type in Message Pack or JSON, either a string, a two element array or a JSON object or Message Pack map (referred to as object/map in the rest of this specification). When Transit data is read, any encoded values are decoded and programming-language appropriate representations are produced. Transit defines the rules for encoding and decoding semantically typed values. It does not define how encoded data is stored, transmitted, or otherwise used.

There are two write modes for JSON –

In normal JSON mode, caching is enabled and maps are represented as arrays with a special marker element. There is also JSON-Verbose mode, which is less efficient, but easier for a person to read. In JSON-Verbose mode, caching is disabled and maps are represented as JSON objects. This is useful for configuration files, debugging, or any other situation where readability is more important than performance. A JSON reader is expected to transparently handle data written in either mode and to remain unaware of which mode was used to write the data.

II. JSON

JSON is a data interchange format which is really simple, it has a self-documenting format, it is much shorter because there is no data configuration overhead. That is why JSON is considered a fat-free alternative. Though it is one of the most used data interchanged format, there is still room for improvement. For instance, JSON uses excessively quotes and key names are very often repeated. This problem can be solved by JSON compression algorithms. There are more than one available.

Example of JSON

```
{
“book”:  [
{
“id” : “01”
“language” : “java”,
“edition”: “third”,
“author”: “Herbert Schildt”,
},
{
“id” : 07”
“language” : “C++”,
“edition”: “second”,
“author”: “E.Balagurusamy”,
}
]
}
```

JSON is built on two structures:

A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON's basic Data Types, Syntax and Example :

Number: a signed decimal number that may contain a fractional part and may use exponential E notation, but cannot include non-numbers like NaN. The format makes no distinction between integer and floating-point. JavaScript uses a double-precision floating-point format for all its numeric values, but other languages implementing JSON may encode numbers differently.

String: a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.

Boolean: either of the values true or false.

Array: an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation with elements being comma-separated.

Object: an unordered collection of name/value pairs where the names (also called keys) are strings. Since objects are intended to represent associative arrays. it is recommended, though not required, that each key is unique within an object. Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.

Null : An empty value, using the word null

Whitespace is allowed and ignored around or between syntactic elements (values and punctuation, but not within a string value). Four specific characters are considered whitespace for this purpose: space, horizontal tab, line feed, and carriage return. JSON does not provide any syntax for comments.

Early versions of JSON (such as specified by RFC 4627) required that a valid JSON "document" must consist of only an object or an array type, which could contain other types within them. This restriction was removed starting with RFC 7158, so that a JSON document may consist entirely of any possible JSON typed value.

JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An object is an unordered set of name/value pairs. An object begins with **{** (left brace) and ends with **}** (right brace). Each name is followed by **:** (colon) and the name/value pairs are separated by **,** (comma).

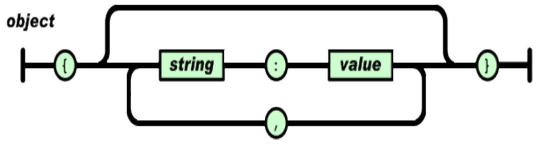

Figure (a) Object in JSON

An array is an ordered collection of values. An array begins with **[** (left bracket) and ends with **]** (right bracket). Values are separated by **,** (comma).

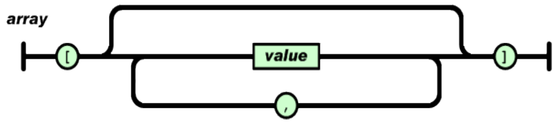

Figure (b) Array in JSON

A value can be a *string* in double quotes, or a number, or true or false or null, or an object or an *array*. These structures can be nested.

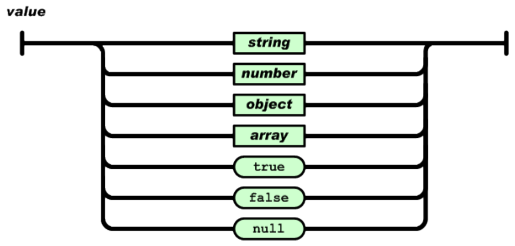

Figure (c) Value in JSON

A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.

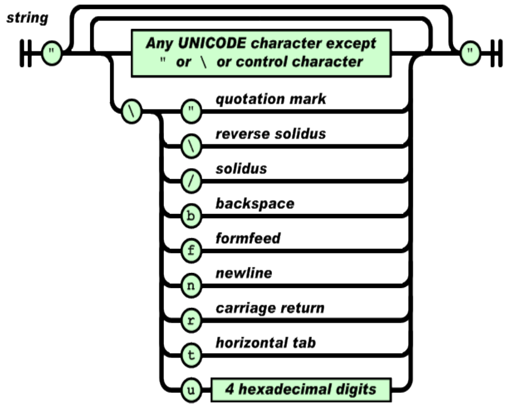

Figure (d) String in JSON

A *number* is very much like a C or Java number, except that the octal and hexadecimal formats are not used.

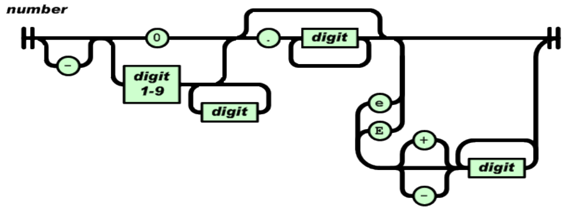

Figure (e) number in JSON

Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, that completely describes the language.

III. PROBLEM DEFINITION

As web services performances are imperative and plays an eminent role in data transmission over the network, We need more faster mediation services to improve the performance which are faster , secure and simple to understand.

IV. PROPOSED TECHNIQUE

Compressed JSON and its Features

JSON used to have an advantage because it could be directly parsed by a java script engine, but even that advantage is gone because of security and interoperability concerns. Compression of JSON data is useful when large data structures must be transmitted from the web browser to the server. In that direction, it is not possible to use gzip compression, because it is not possible for the browser to know in advance whether the server supports gzip. The browser must be conservative, because the server may have changed abilities between requests.

FEATURES:

I. Serialization

When using JSON, we see that the frameworks reduce the serialization time drastically. JSON serialization appears to give us an gain of 50-97% in serialization time.

II. Data Storage

From the data storage aspect, almost every data interchange format take similar space such as Xml , JSON However, Xml based *string* definitely requires more storage space. So if it comes to storing string, JSON is the clear choice at benefit JSON still saves some bytes and this is one reason some NoSQL databases uses JSON based storage instead of XML based storage. However, for quicker retrieval you need to apply some indexing mechanisms too.

III. Data Transfer

Data transfer comes in picture when you are transferring your objects on EMS / MQ / Web Services. Keeping other parameters such as network latency, availability, bandwidth, throughput, etc. as constants in both cases amount of data transfer becomes a function of data length or protocol used over network. For EMS / MQ – Data length, as in statistics, is lesser in case of JSON when sent asstring and almost same when sending as compressed bytes.ENCRYPTION

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption does not of itself prevent interception, but denies the message content to the interceptor. In our work , We will use an encryption Technique to encrypt and decrypt the data before sending and receiving the data over the Internet.

Compared Algorithms for Encryption for compressed JSON

DES: (Data Encryption Standard), The DES was once a predominant symmetric-key algorithm for the encryption of electronic data. But now it is an outdated symmetric key data encryption method. DES uses 56 bits key for encryption and decryption. It completes the 16 rounds of encryption on each 64 bits block of data.

3DES: As an enhancement of DES, the3DES (Triple DES) encryption standard was proposed. In this standard the encryption method is similar to the one in original DES but applied 3 times to increase the encryption level. But it is a known fact that 3DES is slower than other block cipher methods. Encryption strength is directly tied to key size, and 56-bit key lengths have become too small relative to the processing power of modern computers. So, 3DES is simply the DES symmetric encryption algorithm, used three times on the same data. Three DES is also called as T-DES. It uses the simple DES encryption algorithm three times to enhance the security of encrypted text.

AES: (Advanced Encryption Standard), is the new encryption standard.AES is actually, three block ciphers, AES-128, AES-192 and AES-256. Each cipher encrypts and decrypts data in blocks of 128 bits using cryptographic keys of 128 bits, 192 bits and 256 bits, respectively. In Advanced encryption standard there are 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys. Brute force attack is the only effective attack known against it, in which the attacker tries to test all the characters combinations to unlock the encryption. Both AES and DES are block ciphers.

Blowfish: It is one of the most common public domain encryption algorithms, Blowfish is a variable length key, 64-bit block cipher. The Blowfish algorithm was first introduced in 1993.This algorithm can be optimized in hardware applications though it's mostly used in software applications. Though it suffers from weak keys problem, no attack is known to be successful against. It operates on block size 64 bits. It is a 16-round Feistel cipher and uses large key dependent S-Boxes. Each S-box contains 32 bits of data.

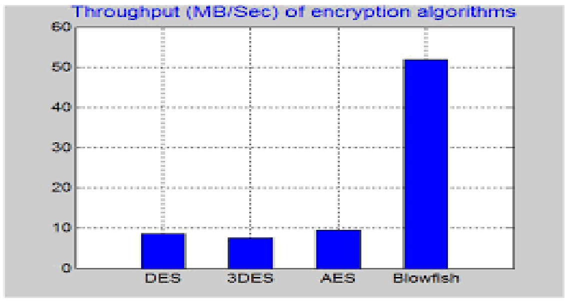

Figure (f)

Figure(f) compares all four algorithms throughput parameter in terms of performance. Throughput is calculated as request per unit time.

Payload Data	Throughput (Mb/Sec)			
	3DES	DES	AES	Blowfish
10	20	40	60	100
20	50	70	90	180
30	70	90	100	260
40	90	110	130	320
50	100	130	180	440
60	130	150	270	550
70	160	200	380	700

Table 1

Table 1 shows the difference in throughput(Mb/Sec) when different payload data is applied to the given encryption algorithms and Blowfish efficiency in terms of throughput.

Compressing JSON with CJSON algorithm

CSJON compress the JSON with automatic type extraction. It tackles the most pressing problem: the need to constantly repeat key names over and over. Using this compression algorithm, the following JSON:

```
[
{ // This is a point
   "x": 100,
   "y": 100
 }, { // This is a rectangle
   "x": 100,
   "y": 100,
   "width": 200,
   "height": 150
 },
 {}, // an empty object
]
```

Can be***compressed***as:

```
{
 "templates": [
   [0, "x", "y"], [1, "width", "height"]
 ],
 "values": [
{ "values": [ 1,  100, 100 ] },
{ "values": [2, 100, 100, 200, 150 ] },
   {}
 ]

}
```

Compressed JSON removes the key: value pair of json's encoding to store keys and values in separate parallel arrays:

```
//uncompressed JSON

JSON = {

Data : [

    {   field1 : 'data1', field2 : 'data2', field3 : 'data2' },

{   field1 : 'data4', field2 : 'data5', field3 : 'data6' },

.....

]

};

//compressed JSON

JSON = {

   Data : [ 'data1', 'data2', 'data3', 'data4', 'data5', 'data6' ],

Keys : [ 'field1', 'field2', 'field3' ]

};
```

More Reduction

If the field is repeated very often and it is a string type, you can get compressed a little be more if you add a distinct list of that field... for instance, a field name job position, city, etc are excellent candidate for this. You can add a distinct list of this items and in each item change the value for a reference number. That will make JSON more lite.

PERFORMANCE ANALYSIS AFTER LOAD TESTING

As we created services and performed load testing using Apache Jmeter tool, and the graph represents the major difference between Optimized JSON and XML as Optimized JSON is taking less time compare to XML

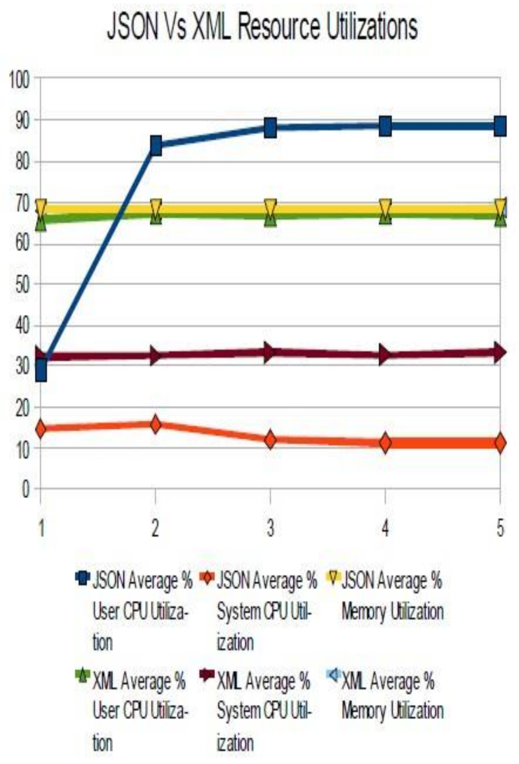

Figure (g)

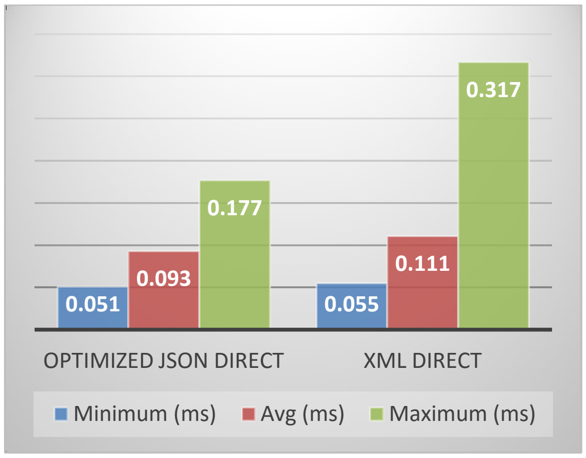

Figure (h)

Figure (h) explains the time comparison between optimized JSON Direct and XML Direct for minimum, average and maximum time milliseconds.

V. CONCLUSION

The proficiency of the proposed mediation services has been tried regarding performance analysis of optimized services in terms of simplicity and calculating time. The outcome demonstrates a surprising change as far as data storage capacity and altogether lessened calculation time. The results are better than all other well known algorithms proposed. Summing up the proposed algorithm is a better replacement of the entire group of compared algorithm in terms of comprehensibility, reduced complexity, reduced computation time and acceptable indeed better than other data interchange formats.

REFERENCES

[1] Extensible markup language (xml) 1.0 (fourth edition). W3C, 2006. http://www.w3.org/TR/2006/REC-xml-20060816

[2] Alexander (2007). "JSON Pros and Cons ". Retrieved April 25, 2012,fromhttp://myarch.com/json-pros-and-cons.

[3] Sporny, M. (2010). "Web Services: JSON vs. XML." Retrieved June 02,2012, from http://digitalbazaar.com/2010/11/22/json-vs-xml/.

[4] RajdeepBhanot and Rahul Hans, A Review and Comparative Analysis of Various Encryption Algorithms, International Journal of Security and Its Applications Vol. 9, No. 4 (2015), pp. 289-306.

[5] E. Biham and A. shamir, "A differential cryptoanalysis of data encryption stamdard", Springer-verlag, (1993).

[6] B. Schneier, “Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)”, [online] Available at: http://www.schneier.com/paper-blowfishfse.html.

[7] S. Basuin, “International data encryption algorithm (idea) – a typical illustration”, Journal of global research in computer science (JGRCS), vol. 2, no 7, (2011).

[8] A. Kakkar and M. L Singh, “Comparison of Various Encryption Algorithms and Techniques for Secured Data Communication in Multinode Network”, Published in International Journal of engg, and technology(IJET), vol. 2, no. 1, (2012).

[9] J. Daemen, R. Govaerts and J. Vandewalle, “Weak Keys for IDEA”, Springer-Verlag, (1998).

[10] M. Abutaha, M. Farajallah, R. Tahboub and M. Odeh, “Survey Paper: Cryptography Is the Science of Information Security”, published in International Journal of Computer Science and Security (IJCSS), vol. 5, no. 3, (2011).

[11] GarimaDutt, Anup S. Kushwaha, Review Of JSON Data Interchange Format, published in International Journal Of Innovative Research In Technology (IJIRT),Volume 3 Issue 1 June 2016 .

[12] JSON. json.org. http://www.json.org

[13] Querying JSON with Oracle Database 12C, Oracle White Paper, July 2015.

[14] Bruno Gil, P. T. (2011). Impacts of data interchange formats on energy consumption and performance in smart phones. OSDOC '11b Proceedings of the 2011 Workshop on Open Source and Design of Communication, NY, USA ACM.

[15] C. Zakas, N. M., J. and Fawcett, J. (2006). Professional AJAX, University of Huddersfield.

[16] EICHORN, J. (2006). Understanding AJAX United States: prentice hall, University of Huddersfield.

[17] Esposito, D. (2007). Introducing Microsoft ASP.NET AJAX Microsoft Press, University of Huddersfield

Iftikhar Ahmad, A. A., Abdullah SharafAlghamdi (2010). "Evaluatingn Intrusion Detection Approaches Using Multi-criteria Decision Making Technique." International Journal of Information Sciences &