Performance Model of Object Serialization using GZip Compression Technique with XML and JSON Formatters

Pooja Manocha[1] Rahul Kadian[2]
[1]M.tech Scholar [2]Assistant Professor
[1,2]Department of Computer Science Engineering
[1,2]CBS Group of Institutions, Jhajjar, Haryana

Abstract— Object Serialization Methods can be useful for several purposes, including object serialization Minimization which can be used to fall the size of serialized data. We have implemented means by serialization and de-serialization of object that can be done using modern format XML and JSON after adding compression to the object streams. Serialization is the process of converting complex objects into stream of bytes for storage. De-serialization is its reverse process that is unpacking stream of bytes to their original form. It is also known as Pickling, the process of creating a serialized representation of object. Object serialization has been investigated for many years in the context of many different distributed systems. Serialization is a process of converting an object into a stream of data so that it can be easily transmittable over the network or can be continued in a persistent storage location. This storage place can be a physical file, record or set of connections Stream.

Key words: Object Serialization, Compression Techniques, Object oriented design, Performance Analytics, JSON DataContract, JSON.NET

I. INTRODUCTION

Object serialization is the ability of an object to write a complete state of it-self and of any objects that it refers to output stream, so that it can be re-established from the serialized representation at a later time [18]. It is also known as Pickling [11], the process of creating a serialized representation of object.

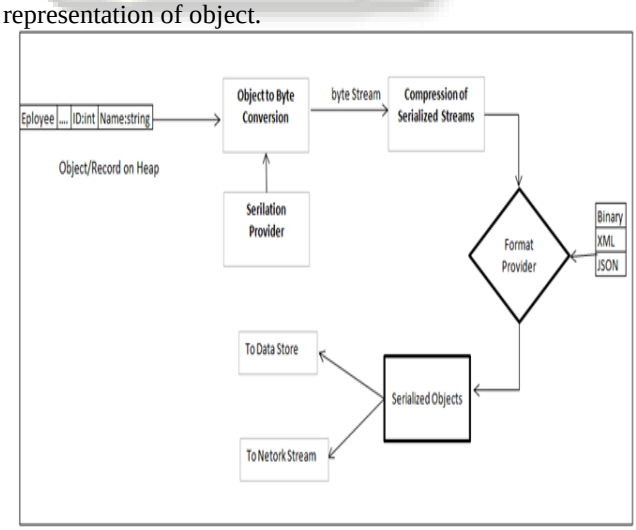

Fig. Compressed Stream Object Serializtion Schematic

Fig. 1: compressed stream object Serialization scheme

During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, is converted to a stream of bytes, which is then written to a data stream. When the object is later de-serialized, an exact replica of the original object is created .Object serialization has been investigated for many years in the context of many different distributed systems.

When implementing a serialization mechanism in an object-oriented environment, users have to create a number of tradeoffs between ease of use and flexibility. The process can be computerized to a large extent, provided the user is given with adequate control over the process. For example, situations may take place where only binary serialization is not enough, or there might be a specific reason to decide which fields in a class need to be serialized.

A. Need for Serialization Process

A concrete example would be a project, where serialization is needed is while storing information from an address book, in this case written in any language as long as it supports Serialization. Every instance may contain a person with details about their address and phone number. One wants to store all instances on a server in exactly the way they are created and there are a few possible solutions;

- The Serialization can easily be done, but problems arise if the data would have to be accessible to applications written in C++, C#, JAVA [4] or an additional language as the data is serialized in a manner unique to that particular language.
- By using an improvised way of encoding the data into single strings, such as encoding four integers into for example 112:93:13:11. This solution requires some custom parsing code to be written, and is most efficiently used when converting very simple data.
- By serializing the data into XML [4]. It is a smart method due to the fact that XML is human readable and have bindings (API libraries) for many languages, although it is space intensive and can cause performance penalties on applications.
- Serialization is often used when transmitting data, as has been mentioned above. Some other examples of such cases are; when storing user preferences in an object or for maintaining security information across pages and applications. When objects are transferred among applications, or through firewalls, serialization can be very helpful.

B. Aplications of Serialization

- A technique for remote procedure calls, e.g., as in SOAP [7].
- A method for distributing objects, particularly in software components such as COM, CORBA [13], etc.
- A method for detecting modifications in time-varying data.

For some of these features to be helpful, architecture independence must be maintained. For example,

for maximum use of distribution, a system working on different hardware architecture should be able to reliably reconstruct a serialized data stream.

This means that the simpler and faster procedure of directly copying the memory layout of the data structure cannot work reliably for all architectures. Inherent to any serialization method is that, because the encoding of the data is by characterization serial, extracting one element of the serialized data structure requires that the entire object be read from beginning to end, and reconstructed. In many applications this linearity is a quality, because it enables simple, familiar I/O interfaces to be utilized to hold and pass on the state of an object. In those applications where higher performance is a concern, it can make sense to burn up more effort to deal with a more composite, non-linear storage organization.

C. Drawbacks of Serialization Process

Serialization, however, breaks the opacity of an abstract data type by potentially exposing private implementation details. Trivial implementations which serialize all data members may violate encapsulation.

To discourage competitors from making well-suited products, publishers of proprietary software often keep the details of their programs' serialization formats a trade secret. Some deliberately complicate or even encrypt the serialized data. Yet, interoperability requires that applications be able to understand each other's serialization formats. Therefore, remote method call architectures such as CORBA define their serialization formats in detail.

Many people attempt to future proof their backup archives—in particular, database dumps—by storing them in some relatively human-readable serialized format.

D. Data Compression

Data Compression [16] or bit-rate reduction involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and removing statistical copying.

No information will vanish in lossless compression. Lossy compression reduces bits by identifying avoidable information and removing it.

Compression is useful because it helps reduce resources requirement, such as data storage space or transmission capacity.

Data compression is subject to a space-time complexity trade-off [17]. For example, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed, and the alternative to decompress the video in full before watching it may be inconvenient or require extra storage. The plan of data compression schemes involves trade-offs among various factors, including the amount of compression, the amount of distortion introduced (*e.g.*, when using lossy data compression), and the computational assets required to compress and uncompress the data.

II. THEORETICAL FOUNDATION OF RESEARCH

Any serialized representation of an object should have the following capabilities: It should be platform and language independent, since

- Serialization and de-serialization could be carried out on different platforms.
- Its validity must be easily verified.
- It should be simple to de-serialize.
- Currently there is much effort going on in using XML, JSON as a means of serializing objects. The following research areas can be distinguished: serializing .NET objects, serializing data from file into XML and JSON Format.

Object Serialization has been studied exclusively on Java Platform, however, in most recent Platforms such as .NET this topic is still lagging behind. Microsoft .Net platform provides means for normal Object Serialization. However The research is done on the following areas:

(1) Implement means by which Serialization and De-serialization of Objects can be done over .Net CLR Platform to modern formats XML and JSON.
(2) Implement Compression in Object Serialized Streams for more efficient Serialization and De-serializtion of objects.
(3) Implement Compressed Object Serialized Streams that can be used for Serialization to any medium Binary, XML, JSON.
(4) It also measures the comparitive Performance of Object Serialization in a Normal CLR Binary and XML and different Types of JSON formatters.

III. PRACTICAL IMPLEMENTATION AND RESULTS

Performance of the Object Serialization is measured on the basis of the comparative analysis of the three types of formatters.

A. Binary Serialization

Binary Serialization is a mechanism which writes the data to the output stream such that it can be used to re-construct the object automatically. The term binary in its name implies that the necessary information that is required to create an exact binary copy of the object is saved onto the storage media. A notable difference between Binary serialization and XML serialization is that Binary serialization preserves instance identity while XML serialization does not.

B. XML Serialization

XML serialization converts (serializes) the public fields and properties of an object or the parameters and returns values of methods, into an XML stream that conforms to a specific XML. The serialization of in-memory object instances of a class into corresponding XML documents heavily influences the performance of the XML-based communication, even if we send the XML over HTTP as in the case of SOAP-based XML Web Services, or saving it into a file.

C. JSON Serialization

JSON (JavaScript Object Notation) is an efficient data encoding format that enables fast exchanges of small amounts of data between client browsers and AJAX-enabled Web services [13]. The format has grown to be very popular in cases where serialization and interchange of structured data over network and is often associated with the modern web due to the fact that it is frequently used when communication between a web server and client side web application is requested.[3].

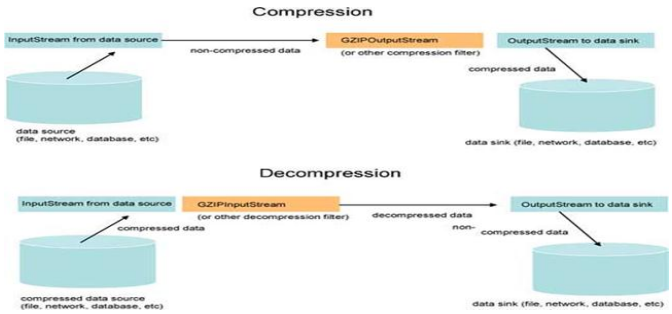

Fig. 2: Compression and Decompression of Serialized Object Streams using Gzip Method

D. Code Snippets and Output

The sample code for JSONDataContract Serialization using GZIP compression technique as well as uncompressed mode.

```
DataContractJsonSerializer jsonSerializer = new DataContractJsonSerializer(typeof(T));
GZipStream compressor;
//Open the file written above and read values from it for Compressed and Uncompressed mode.
if(Compressed)
{
 compressor = new GZipStream(File.OpenWrite(filename), CompressionMode.Compress);

jsonSerializer.WriteObject(compressor, value);
compressor.Close();
}
else
{
Stream stream = new FileStream(filename, FileMode.Create);

jsonSerializer.WriteObject(stream, value);
stream.Close();
}
```

Fig. 3: Sample DataContractJSON Serialization using GZip Method

The sample code for JSON.Net Serialization using GZIP compression technique as well as uncompressed mode.

```
GZipStream compressor;
Newtonsoft.Json.JsonSerializer jsonserializer = new Newtonsoft.Json.JsonSerializer();
if (Compressed)
{
 Using (compressor= new GZipStream(File.OpenWrite(filename), CompressionMode.Compress))
 {
Newtonsoft.Json.JsonConvert.SerializeObject(jsonvalue);
 }
 compressor.Close();
}
else
{
using(StreamWriter stream = new StreamWriter(filename))
 {
 Newtonsoft.Json.JsonTextWriter writer = new Newtonsoft.Json.JsonTextWriter(stream);
jsonserializer.Serialize(writer,jsonvalue);
stream.Close();
writer.Flush();
 }
}
```

Fig. 4: Sample JSON.NET Serialization using GZip Method

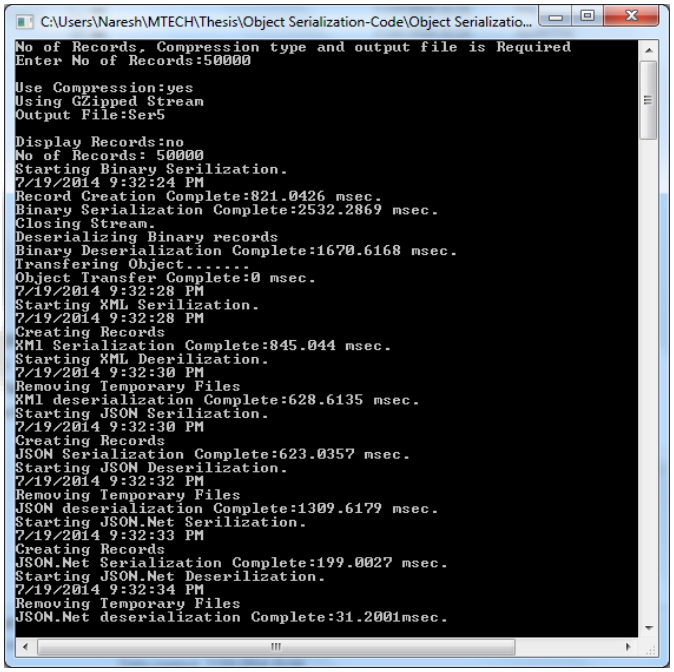

Fig. 5: Generation of 50,000 Records with Compression.

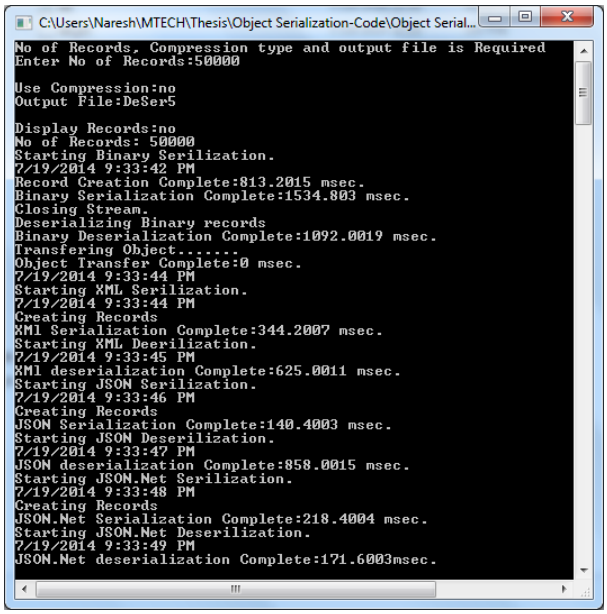

Fig. 6: Generation of 50,000 Records without Compression.

E. Tables and Graphs

Binary Serialization (msec)	Binary Deserialization(msec)	XML Serialization (msec)	XML Deserialization (msec)	JSON Serialization (msec)	JSON Deserialization (msec)	JSON Deserialization (msec)	JSON Deserialization (msec)
648.4076	307.0176	424.0243	194.2007	124.8002	265.2004	189.0065	15.0008
1205.858	608.4011	464.4231	229.0131	220.4049	499.2009	160.604	15.0008
1596.203	765.4014	577.201	328.6006	375.4007	780.0013	187.2004	20.6003
2398.805	1315.403	717.6012	580.2012	532.4011	1142.802	187.2003	15.6001
2532.287	1670.617	845.044	628.6135	623.0357	1309.618	199.0027	31.2001
2907.606	1734.603	1016.002	827.8015	748.8014	1593.203	405.6007	31.2001
4078.608	2816.005	1138.802	828.8016	844.4016	1953.004	252.0144	17.001
4671.409	3157.206	1905.204	1233.402	876.6017	3466.206	327.6006	15.6
4146.211	3188.807	1430.849	1296.802	1026.421	2349.242	463.4014	15.6001
4664.611	3553.263	1635.843	1254.619	1188.602	2685.007	378.4009	15.6

Table 1: Binary XML and JSON Serialization and De-serialization Time with Compression.

Binary Serialization (msec)	Binary Deserialization (msec)	XML Serialization (msec)	XML Deserialization (msec)	JSON Serialization (msec)	JSON Deserialization (msec)	JSON Deserialization (msec)	JSON Deserialization (msec)
405.6008	249.6004	405.6007	78.0001	46.8	202.8004	171.6003	93.6002
814.2016	484.6009	265.2005	111.2003	78.001	296.4005	187.2003	93.6002
891.201	483.6008	280.8005	234.0004	124.8002	686.4012	187.2004	109.2002
1334.0763	1095.0626	361.0206	277.0158	91.0052	607.0347	189.0108	148.0084
1534.803	1092.0019	344.2007	625.0011	140.4003	858.0015	218.4004	171.6003
1671.203	1148.2207	500.2009	825.8273	124.8003	1334.0028	259.6054	202.8003
2719.4051	2013.4035	436.8008	746.2063	202.8003	1265.6024	249.6004	249.6005
2287.4581	2233.8041	463.0265	723.2093	159.0091	1282.405	464.0265	405.6007
2414.138	2206.1262	547.0313	622.0356	179.0103	1376.0787	288.0165	328.0188
2614.1495	2427.1388	545.0312	701.0401	195.0111	1608.092	305.0174	387.0222

Table 2: Binary XML and JSON Serialization and De-serialization Time without Compression

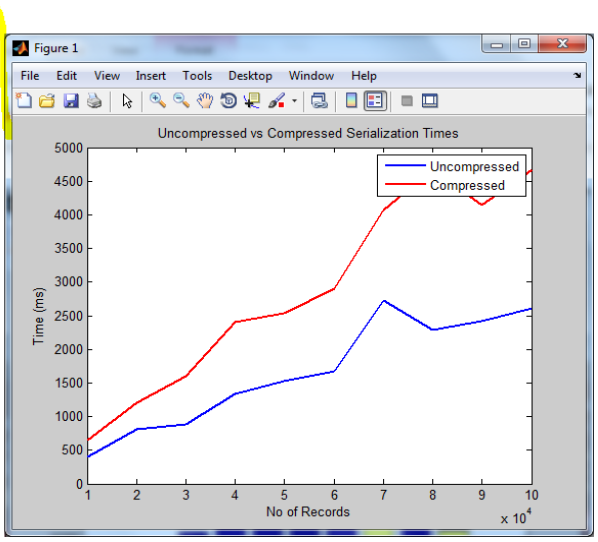

Fig. 7: Binary Serialization

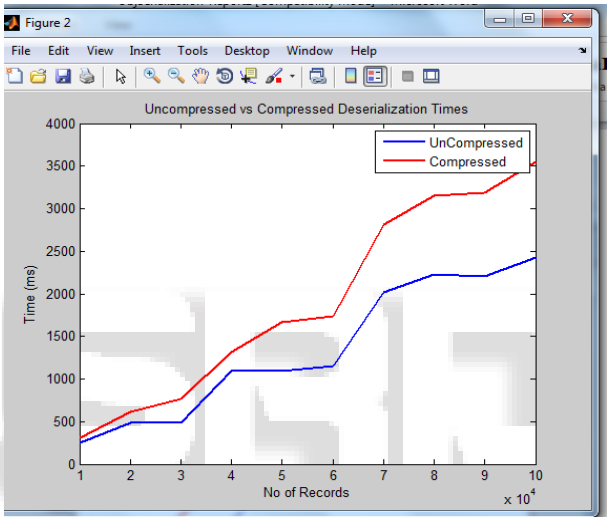

Fig. 8: Binary Deserialization

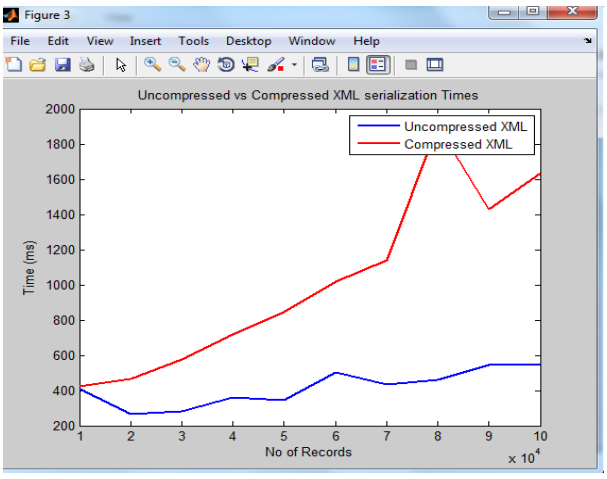

Fig. 9: XML Serialization

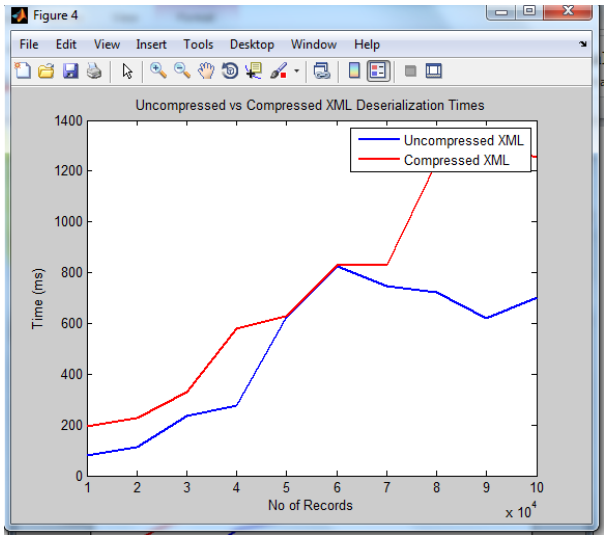

Fig. 10: XML Deserialization

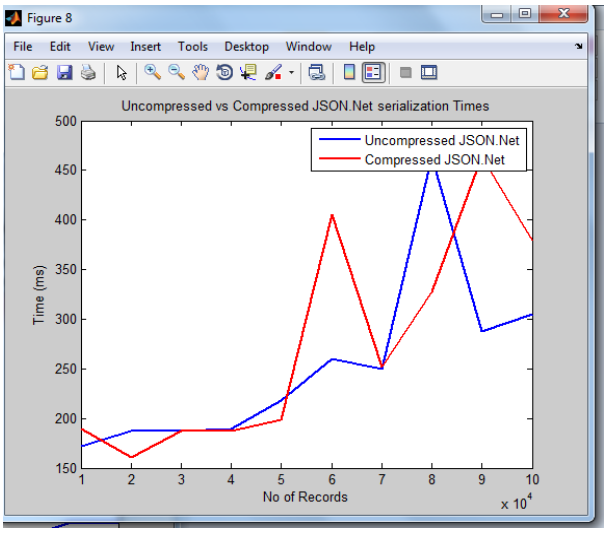

Fig. 13: JSON.NET Serialization

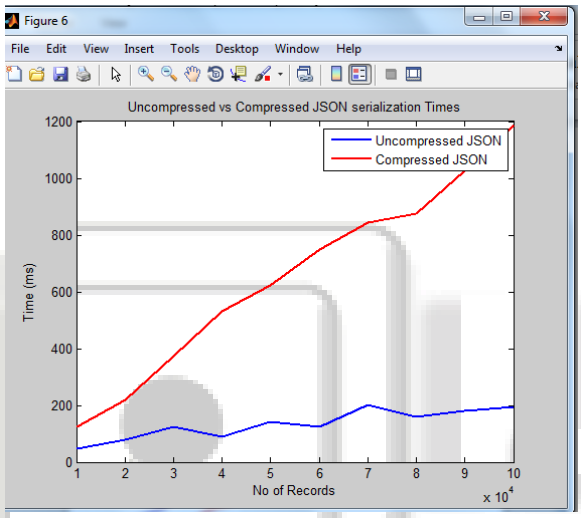

Fig. 11: JSON Data Contract Serialization

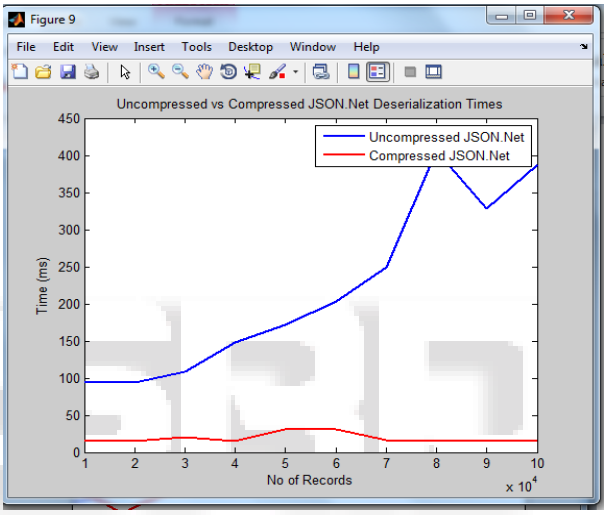

Fig. 14: JSON.NET De-serialization

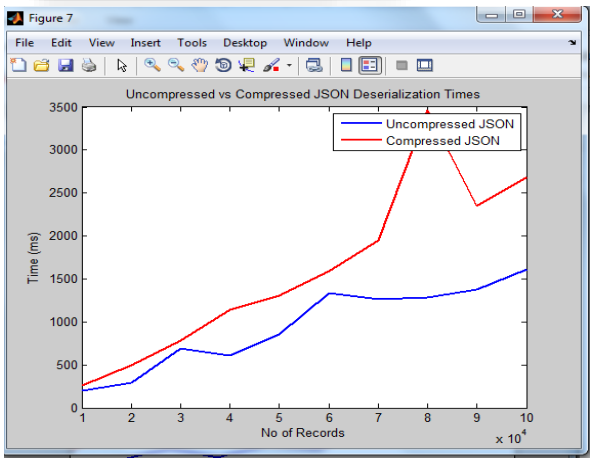

Fig. 12: JSON Deserialization

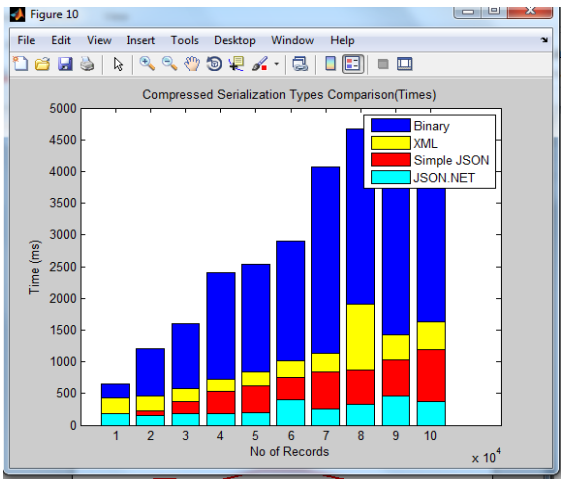

Fig. 15: Time Comparison of various formatters

IV. CONCLUSION

Serialization is a process of converting an object into a stream of data so that it can be easily transmittable over the network or can be continued in a persistent storage location. This storage location can be a physical file, database or Network Stream. This thesis concludes the work that is going on in the field of Object Serialization.

The primary design goals for Serialization, to provide a simple and effective data exchange, but also being easy to generate and load. It is widely used and is used natively available in the most common modern programming. Object Serialization is especially well suited for functional programming languages, where the closure semantics and ability to serialize code are essential. Also a minimization technique helps reduce Serialization sizes considerably.

Object Serialization was studied exclusively on Java Platform, however, in most recent Platforms such as .NET this topic was still lagging behind.

This paper presented Object Serialization Techniques that can be useful for various purposes under .Net Framework, including object serialization Minimization which can be used to decrease the size of Serialized data. We have also formulated statistics for both Compressed and Uncompressed Object Serialized streams using compression techniques viz. **GZip** and serializing formatters i.e. Binary, XML, DataContract JsonSerializer and JSON.NET.

(1) Both serialization and De-serialization show an almost linear relationship for binary serialization with or without compression.
(2) Compressing XML data Increases serialization time exponentially as Records grow, as Compression and Decompression overhead is very large.
(3) For Moderate no of records XML Serialization is a good choice but for large records which require compression binary Serialization however is a better choice.
(4) Compressed JSON Serialization has been used as proposed solution to overcome the above mentioned shortcomings of the XML Serialization.
(5) Test has been performed on the Compressed and Uncompressed serialization and deserialization of various JSON formatter and concluded that Compressed JSON.NET is now 3x times faster than XML and Binary Serialization and also comparatively faster than both the JavaScriptSerializer and the WCF DataContractJsonSerializer over all scenarios.
(6) It has been concluded from my tests and experiment that JSON is smaller in size when compared to equivalent XML and Binary so it is a good choice if no of records are very large.

V. Future Scope

Object serialization is the ability of an object to write a complete state of itself and of any objects that it references to an output stream, so that it can be recreated from the serialized representation at a later time.

Currently there is much effort going on in using modern serialization formats such as XML, DataContractJSON and JSON.NET as a means of serializing objects.

In future we would like to work in following areas:

(1) Object Serialization to be implemented using relational databases into XML and ID Value Pair using JSON and serializing persistent objects from object oriented databases.
(2) Improve XML serialization to reduce object interdependence.
(3) Improve serialization and de-serialization performance using Json.NET over binary (BSON) rather than using current .NET Binary Formatter.
(4) Implement Encoding and decoding of Object Serialized Streams using implemented .Net Formats.

References

[1] Grewal, Mohinder S., and Angus P. Andrews. Kalman filtering: theory and practice using MATLAB. Wiley-IEEE press, 2011.
[2] Orfali, Robert, and Dan Harkey. Client/server programming with Java and CORBA. John Wiley & Sons, Inc., 1998.
[3] MALIN ERIKSSON,VICTOR HALLBERG, "Comparison between JSON and YAML for dataserialization", the School of Computer Science and EngineeringRoyal Institute of Technology, 2011
[4] Imre, G., M. Kaszó, T. Levendovszky, and H. Charaf. "A novel cost model of xml serialization." Electronic Notes in Theoretical Computer Science 261 (2010): 147-162.
[5] G. Imre, et al, "A Novel Cost Model of XML Serialization", Science Direct, Electronic Notes in Theoretical Computer Science, vol 261, Department of Automation and Applied Informatics, Budapest University of Technology andEconomics, Budapest, Hungary, 2010.
[6] Ghassan Samara, Ahmad El-Halabi, and Jalal Kawash. 2007. Compressing serialized java objects: a comparative analysis of six compression methods. In Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology
[7] Box, Don, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. "Simple object access protocol (SOAP) 1.1."
[8] Kwok, Yu-Kwong, and Lap-Sun Cheung. "A new fuzzy-decision based load balancing system for distributed object computing." Journal of Parallel and Distributed Computing 64, no. 2 (2004): 238-253.
[9] Rauschert, Peter, Yuri Klimets, Jorg Velten, and Anton Kummert. "Very fast gzip compression by means of content addressable memories." In TENCON 2004. 2004 IEEE Region 10 Conference, vol. 500, pp. 391-394. IEEE, 2004.
[10] Hericko, Marjan, Matjaz B. Juric, Ivan Rozman, Simon Beloglavec, and Ales Zivkovic. "Object serialization analysis and comparison in java and. net." ACM Sigplan Notices 38, no. 8 (2003): 44-54.
[11] Obermeyer, Piet, and Jonathan Hawkins. "Object Serialization in the .NET Framework." MSDN. http://msdn. microsoft. com/en-us/library/ms973893. aspx (2001).
[12] Park, Jung Gyu, and Arthur H. Lee. "Specializing the Java object serialization using partial evaluation for a faster RMI [remote method invocation]." In Parallel and Distributed Systems, 2001. ICPADS 2001

Proceedings. Eighth International Conference on, pp. 451-458. IEEE, 2001.

[13] Box, Don, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. "Simple object access protocol (SOAP) 1.1." (2000).

[14] N. Bhatti & W. Hassan, et al, "Object Serialization and De-serialization Using XML", Tata McGraw-Hill, ADVANCES IN DATA MANAGEMENT, Vol 1, 2000.

[15] Opyrchal, L.; Prakash, A., "Efficient object serialization in Java," Electronic Commerce and Web-based Applications/Middleware, 1999. Proceedings. 19th IEEE International Conference on Distributed Computing Systems Workshops on , vol., no., pp.96,101, 1999

[16] Held, Gilbert, and Thomas Marshall. Data Compression; Techniques and Applications: Hardware and Software Considerations. John Wiley & Sons, Inc., 1991.

[17] Cleary, John, and Ian Witten. "Data compression using adaptive coding and partial string matching." Communications, IEEE Transactions on 32, (1984): 396-402.

[18] Riggs, Roger, Jim Waldo, Ann Wollrath, and Krishna Bharat. "Pickling state in the Java system." Computing Systems 9, no. 4 (1996): 291-312.

