Serialization/Deserialization and Search using Protocol Buffer and Bloom Filter for High Performance Systems

Rajesh Kumar[1,#], Bhavneesh Tyagi[2]

[1]*Architect – Ericsson R&D, Gurgaon*

[2]*R&D Manager – Ericsson R&D Gurgaon*

#Corresponding author, Email: *choudhary.rajesh123@gmail.com*

***Abstract* – Serialization is the process of converting in memory data structure into a stream of bytes so that it can be stored or transferred where Deserialization is the process of converting the stream of bytes to in memory data structure. Serialization and Deserialization are the basic building blocks for the communication between two entities or designing any key value store. Bloom Filter is a probability-based data structure which is used to determine the existence of an element inside a set of elements. This paper will focus on using Protocol Buffer for serialization/deserialization of data and Bloom Filter for search in performance intensive application like key value store.**

***Keywords*— Protocol Buffer, Bloom Filter, Distributed systems, Serialization and Deserialization.**

1. INTRODUCTION

This paper is proposing the impact of existing serialization/deserialization techniques on standalone as well as distributed systems so that they can cater to demanding performance requirement.

In today's rapidly changing world, performance is the key to ever growing demand of customers as well as vendors like IoT, distributed systems or any standalone application where performance plays a vital role.

If we carefully choose our serialization technique while designing any application, then it will have positive impact on various performance aspects of any application. Some of the aspects are mentioned below

- *Storing Data into Databases or on Hard Drives*: a method which involves converting program objects into byte streams and then storing them into DBs, such as in Java JDBC.
- *Transferring Data through the Wires:* – for instance, web applications and mobile apps passing on objects from client to server and the reverse.
- *Detecting Changes in Time-Varying Data:* – abrupt variations in time series data can represent transitions that occur between states, which is useful for modelling and predicting time series and is found in a variety of application areas. Remote Procedure Call (RPC): It is a protocol, which one program can use to request a service from a program found on another computer on a network without needing to know that network's details. A remote procedure call is also referred to as a function call or a subroutine call.
- *Persisting Data onto Files:* This occurs mainly in language-neutral formats like XML or CSV. Most languages, however, allow the direct serialization of objects into binary using APIs such as the Serializable interface in Java or fstreamclass in C++.
- *Remote Method Invocation (RMI)*: The serialized objects are passed as parameters to functions on a remote machine as if they have been invoked on a local one.
- *Distributing Objects:* A Distributed Object Model. This method is used for instances when programs running on different platforms written in dissimilar languages have to transfer object data over a distributed network using a framework like Common Object Request Broker Architecture (CORBA).

In addition to serialization and deserialization, this paper also focuses on bloom filter while designing a performance intensive application. It is basically a probabilistic data structure to find out the existence of given element in a data set. Please note that there are chances of false positive result. For e.g. Google HBase, Apache Cassandra uses Bloom filters to reduce the disk lookups for non-existent rows or columns.

Avoidance of disk lookups will result in significant impact in performance of database query operation of an application. In addition to that bloom filter will not give false negative for search operation.

2. WHY PROTOCOL BUFFER AND BLOOM FILTER FOR PERFORMANCE INTENSIVE APPLICATIONS.

1) *What is Protocol Buffer:*

Protocol Buffer is a technique to serialize the structured data and it basically facilitates different systems to communicate with each other either by sending byte stream over the wire or storing it.

This technique relies on_interface definition language to have common agreement between the sender and receiver to send and receive the structured data.

2) *How Protocol Buffer works:*

Protobuf has three main components that we need to take care,

- *Message descriptors:* First step in using Protobuf is to define our messages structures in. proto files. It basically contains the schema information with positions about the actual object.
- *Message implementations*: Messages definitions are not enough to represent and exchange data in any programming language. We must generate classes and their respective objects to handle the data in the respective programming language. You can use protocol compiler provided by google to generate the same.
- *Parsing and Serialization*: After defining and creating Protobuf messages, we need to be able to exchange these messages. Google helps us here again if we use one of the supported programming languages.

3) *Performance comparison of Protobuf with other serialization techniques*

It has been observed that protobuf is the most efficient serialization deserialization technique as it is many times faster than well-established techniques like Xstream, Java Serialization etc. When we did the comparison of different serialization and deserialization techniques over a set of data.

If serialization in any format permitted, Protobuf is the best. Please refer to below mentioned table for serialization performance aspect of different protocols.

Experiment: To examine the performance of serialization and deserialization of structured data, an experiment was designed using the following hardware and software [2]:

- Hardware: iMac (by Apple Inc.) with Intel Core2 Duo
- 2.66 GHz and 2 GB memory
- Operating System: Mac OS X version 10.6.8
- Java: version 1.6.0 29
- Current version of object serialization libraries were selected shown in the following.

The experiment was designed as follows:

- Ten kinds of orders were prepared. They were orders with ten sizes of options: 0, 100, 200, 300, 400, 500, 600, 700, 800 and 900 options.
- 500 iterations were executed for warming up, and then 500 iterations were executed for measuring.
- The serialized file size was measured in bytes, and the execution time was measured in System.currentTimeMillis().

Table I: Average execution time for serialization and deserialization protocols [2].

	Serialization Average (ms.)	Deserialization Average(ms.)
XStream in	9.7858	4.2166

JSON		
JsonLib	2.9422	22.112
XStream in XML	2.8698	5.1280
FlexJson	0.9740	1.4978
ThriftJson	0.7422	0.0020
Gson	0.4756	0.3638
JsonMarshaller	0.4094	0.9874
Jsonic	0.3044	0.9038
Object serialization in Java	0.2606	0.2800
JsonSmart	0.2328	0.1710
AvroJson	0.2278	0.4036
Thrift in binary	0.1654	0.1316
Avro in binary	0.1672	0.0996
ProtoStuff with static schema	0.1750	0.1626
ProtoStuff with dynamic schema	0.1532	0.1922
Jackson	0.1488	0.2098
Protobuf	0.0476	0.1058

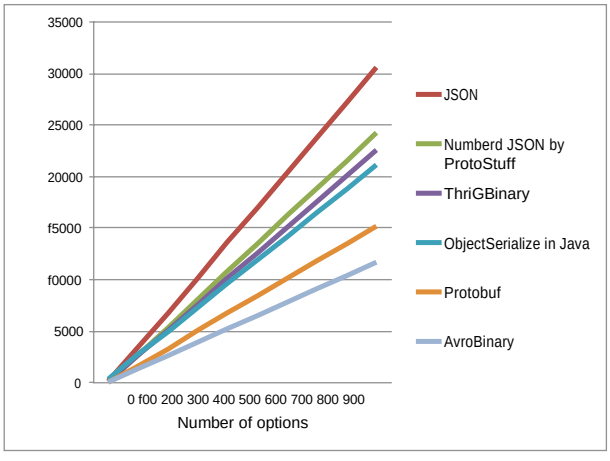

Fig. 1: Size comparison of objects using different serialization protocols [2]. X axis represents number of options and Y axis represents size of objects.

If we refer to serialization and deserialization tables, protobuf [Table. I] seems to be very efficient in compare to other serialization techniques as execution time and size of the serialized object [Fig. 1] is very less and can increase our performance of application by many times provided it involves storing the objects at many places.

4) Security aspect of default java serialization

Hackers and security experts do the investigation of serialization techniques in Java and commonly used third party libraries and look for various methods which can be executed during deserialization to harm various systems.

Attackers can easily execute a denial of service attack by making deserialization process longer using the streams known as deserialization bombs.

In this way, an application or system gets vulnerable to an attack whenever you deserialize a byte stream that you don't trust. The best way to avoid such kind of issue is basically not deserialize anything.

This paper recommends not to use default java deserialization while designing a new system.

There are already various mechanism exists which can be used for object translation between byte sequence and avoid the dangers of java default serialization. In addition to it, these mechanisms also offer various benefits like high performance, cross platform support and nice ecosystem of various tools. In comparison to java serialization, these mechanisms are simpler and don't support serialization and deserialization of arbitrary object graphs. These mechanisms support structured data-objects consisting of a collection of attribute-value pairs. This simple abstraction turns out to be enough for building extremely powerful distributed systems and simple enough to avoid the serious problems that have plagued Java serialization since its inception. Protocol Buffer offered by google is one of the leading cross-platform structured data representation and offers above mentioned benefits.

3. WHY BLOOM FILTER IN PERFORMANCE INTENSIVE APPLICATIONS – SPACE AND TIME ADVANTAGE

Bloom filter is a specific data structure which is

specifically designed to search the existence of an element inside a set while maintain space and time efficiency.

1) *How Bloom Filter works:*

An initial Bloom filter is a bit array of *m* bits, all initially set to 0. There are *k* different hash functions designated, each of which does the mapping or hashing a set element to one of the *m* array positions, resulting in a consistent random distribution. Generally, *k* is a minor constant which rely on the desired false error rate ε, while *m* is proportional to *k* and the number of elements to be added in an array.

Addition of element: To add an element, first it gets inputted to each the *k* hash functions to get *k* array positions. Resulting positions bits gets set to 1.

Search of element: To search an element in a set, input element gets presented against *k* hash functions to get *k* array positions. If any of the position derived after hashing is 0 then it means that element does not exist in the set. If it exists, then the condition is to basically have all bit positions must be set to 1. It might possible that even if all bit position has been found to be 1, element might not be there. This phenomenon is known as false positive in bloom filter.

2) Advantages of Bloom Filter:

Space efficiency: Space efficiency: Bloom filter does not store the actual items. In this way it's space efficient. It's just an array of integers. It also saves expensive data scanning across several servers depending on the use case.

Time efficiency: Bloom Filters have a unique feature which ensures that the time required to insert and search a specific element or item inside the set is fixed constant and this feature is independent of the number of items existed inside the set.

So, if we are designing a performance intensive key value store, then we need to handle lot of search and get queries. If we go by the traditional approach, then it will take hell lot of time and space just to check the presence of an item.

For e.g. if we are having a web URL cache. Rule is to save the results if user had hit a specific URL more than once for the purpose of content delivery. Traditional way of doing it to save all hit URL in some data structure and do the search and count if it is hit more than once. Alternatively, we can use bloom filter to find out in our bloom filter using the bits whether a specific URL exists or not. It will save lot of space and execution time. In a survey it has been found that almost ¾ of URL are hit only once. So, we can imagine how much space and execution time get saved by using the bloom filter.

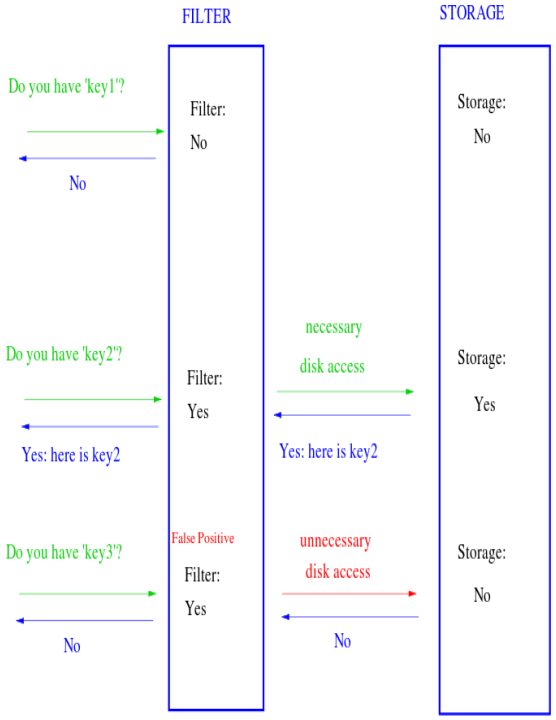

Fig. 2: Bloom filter used to speed up answers in a key-value storage system. values are persisted on a drive which has slow retrieval times. bloom filter decisions are much faster.

3) How to calculate false positive of bloom filter:

There are three factors which decides the positive false rate ***p*** in a bloom filter

- ***m:*** size of the filter.
- ***k:*** number of hash functions.
- ***n:*** number of elements inserted in filter.
- ***p:*** positive false rate of filter.

Above mentioned combination decides the false positive rate *P* by having the equation (1) .

$$P = (1-e^{-km/n})^{k} \quad (1)$$

k is the optimal number of hash function for a bloom filter where ***n*** is size of bit array and ***m*** is the size of the filter depicted in equation (2).

$$k = log(2)(n/m) \quad (2)$$

4) How to decide which bloom filter to use:

There are various types of bloom filter available as per the applicability. Some of these bloom filters are

Stable Bloom Filter: stream of data: Stable bloom filter are the filters which are designed to handle the streaming of data. As streamed data history can be infinite, stable bloom filter continuously remove stale data to make space for more recent data inserted inside the filter. As stale data is removed, it introduces the false negative which does not exist in traditional bloom filter.

Scalable Bloom filter: dynamic data set: This filter can adapt dynamically to the number of elements stored, while assuring a minimum false positive error rate. The technique is based on sequences of standard Bloom filters with increasing capacity and tighter false positive error rate probability, to ensure that a maximum false positive probability can be set in advance, nevertheless of the number of elements to be inserted.

Spatial Bloom filter: location privacy: It is a data structure designed to store location information, especially in the context of cryptographic protocols for location privacy.

Layered Bloom filter: how many times data added: Layered Bloom filters permits keeping history of how many times an item was added to the Bloom filter by checking how many layers contain the item. In this filter, a query operation will generally return the deepest layer number in which the item was found in.

4. CONCLUSION:

The proposed work has focused on designing a performance intensive system in the area of serialization deserialization using protocol buffer and search using bloom filter probabilistic data structure of the stored data.

This paper shows that protocol buffer has very good performance impact in terms of execution time [Table I] and size [Fig. 1] for serialization and deserialization in compare to other existing mechanisms. Protocol Buffer also take care of the security aspect as schema and actual data files are different hence, prevent DOS attacks from tampered serialized objects. There are few constraints like human readability and dynamic schema changes which used needs to look into before using protocol buffer for the application.

Second aspect of this paper focus on considering the bloom filter, if your application has lot of search-based operation where you need to decide the next course of action based upon the existence of some information in your application like key value store.

Usage of bloom filter gives a significant advantage in terms of execution time [Fig. 2] as well as storage as it does not store the actual data, but the bits derived from different hash functions. User can take advantage of different types of bloom filters as per the specific usage [Section 3.4].

Before deciding the bloom filter, rate of false positive should also be considered as there might be some application where false positive rate should be low. In those cases, we need to increase the number of hash functions to decrease the false positive which in turn might decrease the performance of bloom filter.

REFERENCES

[1] Gurpreet Kaur and Mohammad Muztaba Fuad. An Evaluation of Protocol Buffer.
[2] Kazuaki Maeda, Evaluation of Object Serialization Libraries in XML, JSON and Binary Formats.
[3] A. K. Jain and R. C. Dubes. Algorithms for clustering data.
[4] Bojan Mrazovac and Dražen Pezer, "Performance Evaluation of Using Protocol Buffers in the Internet of Things Communication."
[5] Google Protocol Buffer by Google, https://developers.google.com/protocol-buffers/

[6] Carlos Baquero and Nuno Preguic¸, "Scalable bloom filters."
[7] Zhiyang Li, Wenyu Qu and Peng Xiao, "An Efficient DDoS Detection with Bloom Filter in SDN."
[8] Junho Jeong, Jong Wha J, Yangsun Lee and Yunsik Son, "Secure Cloud Storage Service Using Bloom Filters for the Internet of Things."
[9] Apache Thrift, http://thrift.apache.org/.
[10] jsonmarshaller - http://code.google.com/p/jsonmarshaller/.
[11] XStream – About XStream, http://xstream.codehaus.org/.
[12] M. Slee, A. Agarwal and M. Kwiatkowski, "Thrift: scalable cross language services implementation", Whitepaper, Facebook, 156University Ave, Palo Alto, CA.
[13] T. Shuang, T. Lin, L. Xiaoling, and J. Yan, "An efficient method for checking the integrity of data in the cloud."
[14] T. Aditya, P. K. Baruah, and R. Mukkamla, "Space-efficient bloom filters for enforcing integrity of outsourced data in cloud environments."
[15] Tong, Endong, et al. "Bloom filter-based workflow management to enable QoS guarantee in wireless sensor networks."
[16] Zhouguo, Chen, et al. "Design of IP Traceback System based on Generalized Bloom Filter."
[17] Yan, Qiao, et al. "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments."
[18] Bloom Filter, Wikipedia https://en.wikipedia.org/wiki/Bloom_filter/