Poli Data Compressor

Summary:

The present project refers to a programming to be used in databases, computer systems, mobile applications, smart electronic devices and new Blockchains, to generate more space savings on memories, hard drives or any other data storage device, in addition to increasing the speed in data transmission and reception. Ideal for software makers, mobile application developers, and smart electronic device creators (IOT).

The most Blockchains receive a huge load of data and it is believed that in the next years they should receive and keep records of thousands of different information. Currently, there is no specific data packer for this purpose, and the values charged by the miners already consider the number of bytes to be stored.

The system in question will allow energy savings and the prices of the tariffs charged for storage in these types of block databases, precisely by compressing the data before storing them.

The main focus is licensing negotiations for application and software vendors that using Blockchain for data storage. As well as, Blockchains manufacturers and intelligent electronic solutions (IOT).

This system is a set of algorithms, which aims to be used for a compress, according to the type of data that represents the information to be stored.

For a numeric field a specific algorithm is used, for an alphanumeric field another algorithm is used and for a date field a recommendation with numerical algorithm application.

Drawbacks currently observed:

When we talking about data compression, there are several algorithms that follow an arithmetic or byte redundancy coding to perform the compression. These are widely used in file compacts, music, movies, images and there are also those famous files like the MP3 that limit the quality of the sound according to the sensitivity of the human ear.

The most famous are:

- Huffman encoding algorithm that looks for redundancy for compression and is widely used for image compression applications and binary files such as GZIP, PKZIP and BZIP2.
- Roshal File Compression Algorithm that was developed by Eugene Roshal, popularly known as RAR.
- Lempel-Ziv-Welch algorithm which is a compression algorithm created by Terry Welch in the year 1984 and was based on the LZ77 created by Lempel and Ziv. The method uses the dictionary strategy in which each symbol or set of symbols is stored in a file along with its respective index that has a fixed bit value. This fixed value is defined in the LZW algorithm as being 12 bits.

Although these algorithms are widely used, the algorithms that are being proposed through the system in question are intended to pre-identify which bytes are allowed in each field of the form to be submitted to Blockchain and to use encodings that refer to the combinations in a complete byte.

The Project:

Algorithms are available in some common languages on the Internet, as well as the recommended way to develop and adapt to existing codes. In development and testing environments, licenses are not charged, but when

applied to Blockchains, official systems and applications, manufacturers must apply for the exclusive use license for each developed project.

Customers, software and applications makers will receive information on how to use the algorithms through a public internet page and can use the algorithms in their projects at no cost, if they so wish, until the distribution or designalization of the projects on the Internet. During the development period they will be able to check if there are financial gains with the byte compressions, if they can, through the same platform, they can contact by email and request a contract for the release of the license. Thus, it is summarized that the contact platform is the Web and the HTML website system.

These algorithms, for this purpose, compact more than the others in the market because they were made for databases and Blockchain, not for binary generics files, of audio or video. So since there are costs to store by bytes in Blockchains and other databases and these costs are charged because of energy costs to close larger blocks, in the case of Blockchains, there is a decrease in financial costs to customers and a decrease of energy expenditures by miners. Another advantage is in the decrease of the traffic of the information in the web, that once compressed reduces the response time for the applications in the web.

Tests were conducted using Ethereum smart contracts in a blood donor registry project. This project has become the project with the largest amount of web transactions in the world in the RINKEBY test environment (Data collected on 06/14/2018).

After the theoretical definitions focused exclusively on the numerical algorithm, operational tests was registered in the project called, "*Doacao_sangue"*. The purpose is to effectively demonstrate that the costs are higher than the results of the compressed record. Subsequently, other tests were also carried out on this project with a view to better compaction. With respect to the alphanumeric algorithm, a test was carried out in a project of publication of certificates in a closed / private network, in order to analyze costs, response time without the variable (web) and analyzed the compressed size. With this, we conclude that the theoretical variables were well tuned to the operational ones.

Methodology for numbers:

The numbers fields are all found in forms where have just numbers to save, independently of it have a mask or decimals. If we consider that one byte have eight binary combinations we can affirm that is possible to combinate two hundred of fifty and six for byte. So, if we want to save just numbers, ten possibles combinations, we can register it reserving low bits combinations. Supose that we need stock a sequence identified by 255 when masked in a form that we know that have decimals and fix “$”, like $ 2,55. When save it on database, we looking for a byte that represent the combination address 255, binary combination representative [1,1,1,1,1,1,1,1]. If we have another number like 65535 we will find the combination address again and reserve two bytes, identified by 255 and 255.

Methodology for dates:

Certain systems require the completion of dates that do not include hours; for these, we can use the same compression algorithm demonstrated in the numerical methodology, and add the technique of inserting the month followed by the day, composed of two digits, and the year and remove the non-numeric characters of the representation that formats the desired mask. A date like twenty of april of one thousand and nine hundred and seventy six could be represented by 4201976 to be later compressed by the numerical algorithm. This method of reporting the month in the first position allows the compression in the first nine months of the year to be greater than informing it in another position. If you want and know the content of the field, the year represented by only the last two characters can also be used, for example: 42076 to indicate the day April 20, 1976.

Methodology for dates with hours:

If the date is greater than the first day of January 1970, it is recommended to use Unix timestamp formatting before compaction; however, if it is earlier than this date, it is recommended to first use the month digits, then

day with two digits, year with four digits, hour considering twenty-four hours, minutes and last, seconds, if the field requires information of seconds. After this formatting, use the same numerical compression algorithm.

Methodology for hours:

For hours without seconds, it is recommended to use the twenty-four hour pattern followed by the two-decimal minutes. When decoding, first read the last two to consider as minutes and the rest will be the time. To store only hours with minutes and seconds, convert the number of the hour, minutes with the seconds in seconds relative to the day, later, record this number. At seventeen hours and fifty-eight minutes and fifty-nine seconds (17:58:59), we would make 61200 (indicating the seconds of seventeen hours (17 * 60 * 60)) + 3480 (seconds of fifty-eight minutes *(58 * 60)) + 59 (seconds of the fifty-nine seconds) = 64739 seconds. In this way 64739 would be much smaller than if we considered 175859 to apply the compression. Apply the numerical compression.

Methodology for alphanumeric texts:

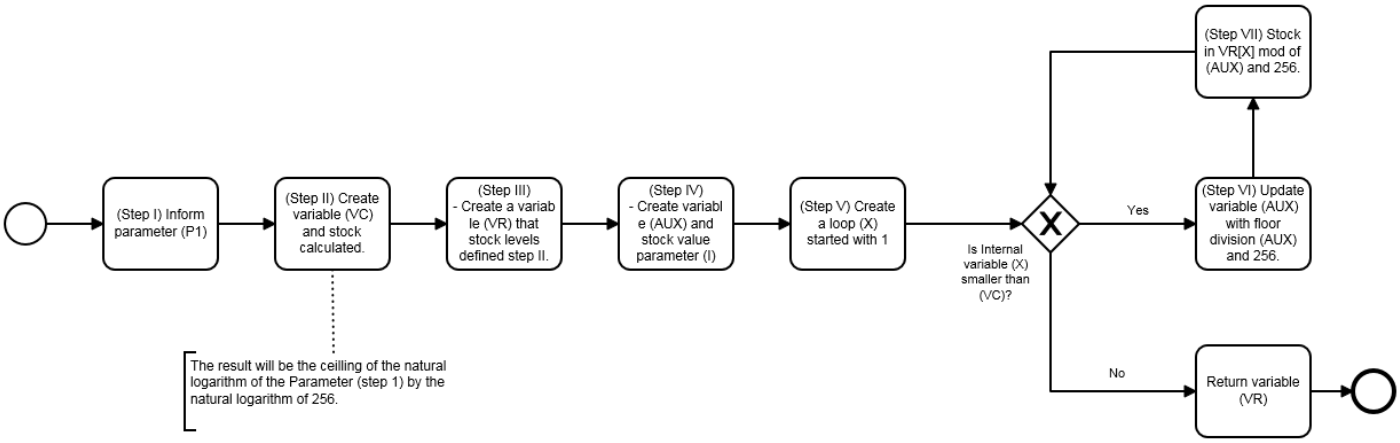
Fields like name, address, email, code and more, combine letters and numbers that will never be added, subtracted, multiplied, and divided. The purpose is to limit what characters the user can use in data entry that is not provided in a list and use the remaining bits of the combinations to store the next bits. Assuming that a person's name is, Marcus Poli, if we consider that the storage will be all uppercase, MARCUS POLI, we will save the lower case letters and when we unpack we will use a treatment to start the first capital letter of the name and surname.

Also in this example, if we consider which numbers are allowed, but accented or non-printable characters should be replaced/denied, we limit the list indicating which characters override the accented characters and we can further compress the field content by removing these characters, however, indicate which letters and numbers should be accepted in this field. As smaller

be the list the greater the compaction power of the algorithm. Fractional and scientific numbers can also be considered as alphanumeric, just treat them in return for decompression.

Formula for numerical compaction:

I. Inform the number you want to compress. (P1)
II. Identify the number of bytes required to compute this number. This information is the result of rounding up to the next integer, when there are fractions, the division of the natural logarithm of the parameter number reported in the previous step by the natural logarithm of two hundred and fifty-six. (VC). If result of (VC) is zero replace to one.
III. Store at the first level of a sequence of bytes with the same amount of levels identified in the previous step, calculating the value module passed by the parameter by two hundred and fifty-six. (VR)
IV. Create a temporary variable and store the number passed by the parameter. (AUX)
V. Create a cursor that starts from one and finishes before reaching the amount of bytes required for the compaction by adding one each time the cursor passes. (X)
VI. While on this cursor, modify the temporary variable (AUX) storage to contain the integer result, neglect decimals, of the previous value divided by two hundred and fifty-six. This will be used at the next cursor level.
VII. Store at the next level of the byte sequence created in step three, the result of the module of the value of the temporary variable (AUX) by two hundred and fifty-six.
VIII. When you finish the cursor, return the values stored in the variable with the byte sequence created in step three. (VR)



Formula for the numerical decompression:

I. Inform the parameter with the sequence of bytes to be unpacked. (P1)

II. Define a variable and assign the byte sequence size of (P1). (VC)

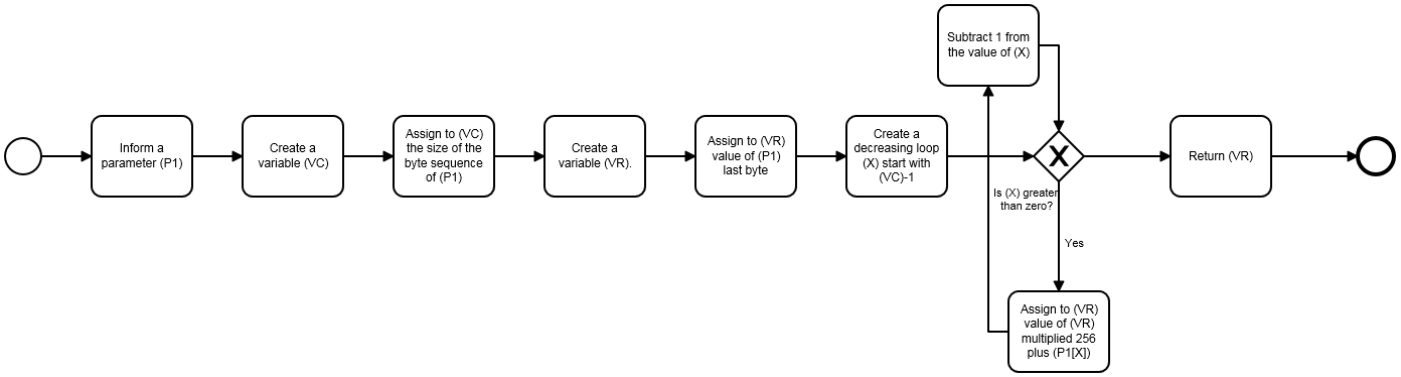
III. Define a numeric variable.

IV. Assign to (VR) the value representing the last byte of the sequence of (P1).

V. Start a cursor with the value of (VC) - 1 and subtract a position of the internal variable (X) with each call of the cursor until it reaches the value 1.

VI. If you are inside the cursor, press (VR) the value of the multiplication of (VR) by 256 and add the result with the value representing the byte of the sequence of (P1 [X]).

VII. When exiting the cursor, return the value of (VR).



Formula for compaction of alphanumeric texts:

I. Load the parameter with the text that you want to compress (PARAM)
II. Create and load a new variable with the list of possible alphanumeric characters (LIST)
III. Load the variable that will identify how many bits are needed to store each of the parameter's characters. (VC). This information is the result of rounding up to the next integer, if there are fractions, of the division of the natural logarithm of the list character count reported in the previous step (LIST) by the natural logarithm of two. If result of (VC) is zero replace to one.
IV. Create a variable that will store a binary sequence of each byte extracted from the (PARAM) parameter. For protection, the same parameter size (PARAM) is maintained. This sequence will identify as (VR).
V. Create a variable (z) and assign zero.
VI. Start a cursor (x) that should find the position of each character of (PARAM) relative to the list informed (LIST). Exit when you find the position of the last text character (PARAM).
VII. Within this cursor (x) load the position that is in the list (LIST) the PARAM character being read at the cursor and assign to a variable here called (aux).
VIII. Start a second cursor (y) that will start at zero and end when it reaches the number of bits, that is, the value of (VC).
IX. Assign to the variable VR, reading the position indicated by (z), the integer value of the module generated by the result of division of (aux) by the result of two raised to position (y) and two.
X. Add one more to the value of the variable (z).
XI. When we finish the two cursors, we will have the result in bits of the compressed sequence, however, we must transform it into bytes again. To do this, we must create a variable that stores a sequence of bytes. This sequence will not be greater than that initiated in the parameters, so we will give the same size. (VRDEC)

XII. Assign zero for two auxiliary variables. (z and aux).

XIII. Start a cursor (x) that will start at zero and end when the content contained in the variable (VR) is finished.

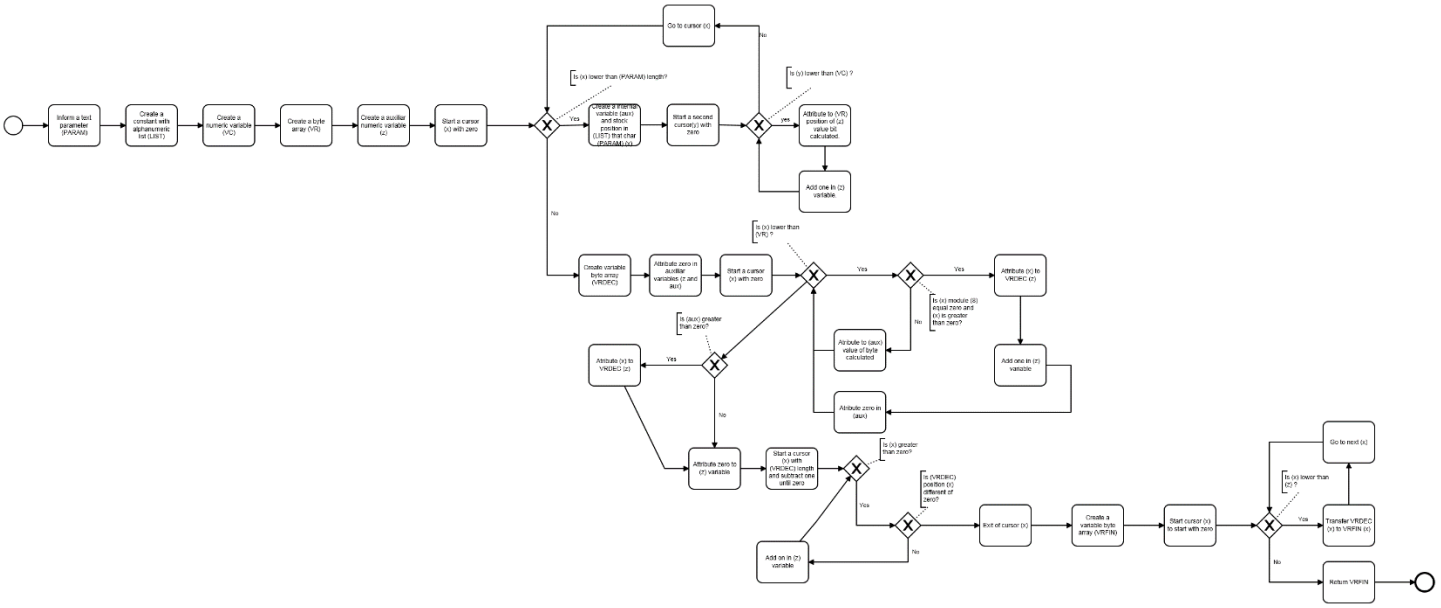
XIV. Within the cursor (x) verify that the value of the modulo of (x) by eight equals zero and the value of the cursor (x) is greater than zero. If the assigned to the variable (VRDEC) of the position (z) the value of the variable (aux); add one more in the variable (z) and assign zero to the variable (aux).

XV. Also, within the cursor (x) assign to the variable (aux) the old value of the variable (aux) plus the result of the multiplication of the value of the variable VR, position (x), by the result of the power of two raised by the module of (x) and eight.

XVI. When leaving the cursor, check if the value of the variable (aux) is greater than zero, if it is, assign to the variable (VRDEC) at position (z) the value of (aux).

XVII. Now we must remove from the sequence of the variable (VRDEC) the decimal values with zeros, leaving the last position until finding an occurrence with a value other than zero. To remove, we will assign zero for the variable (z) and later, we will start reading the variable (VRDEC) from the last position to the first one other than zero.

XVIII. Knowing the number of bytes of the (VRDEC) sequence that we must subtract, we will transfer the bytes to the sequence (VRFIN) which will be the return sequence of the compressed bytes.



Formula for decompression of alphanumeric text:

I. Load the parameter with the binary sequence that you want to unpack. (PARAM)

II. Create and load a new variable with the list of possible alphanumeric characters (LIST)

III. Load the variable that will identify how many bits are needed to store each of the parameter's characters. (VC). This information is the result of rounding up to the next integer, if there are fractions, of the division of the natural logarithm of the list character count reported in the previous step (LIST) by the natural logarithm of two.

IV. Create a variable that will store a binary sequence of each byte extracted from the (PARAM) parameter. For protection, the same parameter size (PARAM) is maintained. This sequence will identify as (VR).

V. Create a numeric auxiliary variable (z) and assign at least one.

VI. Start a cursor (x) that should read all bytes of the binary sequence (PARAM).

VII. Within the cursor (x) start a second cursor (y) which should start at zero and end at seven. This cursor will serve to transform the values of the bytes into bits of (PARAM).

VIII. Add one more in the value of the variable (z).

IX. Assign to (VR) at the position indicated by the variable (z) the result module of the integer value of the division of the value of (PARAM) in the position (x) by the power of two raised by (y) by two.

X. Upon exiting the two cursors, we will have the (VR) sequence filled with bits representing the bytes of the binary sequence of (PARAM). Assign (-1) to the value of the numeric auxiliary variable (z), create a numeric auxiliary variable (v_dec), assign zero to that variable, and create a new Alphanumeric variable to load the return (VRDEC).

XI. Create a cursor (x) that will start from scratch and end when you read all of the bits of (VR).

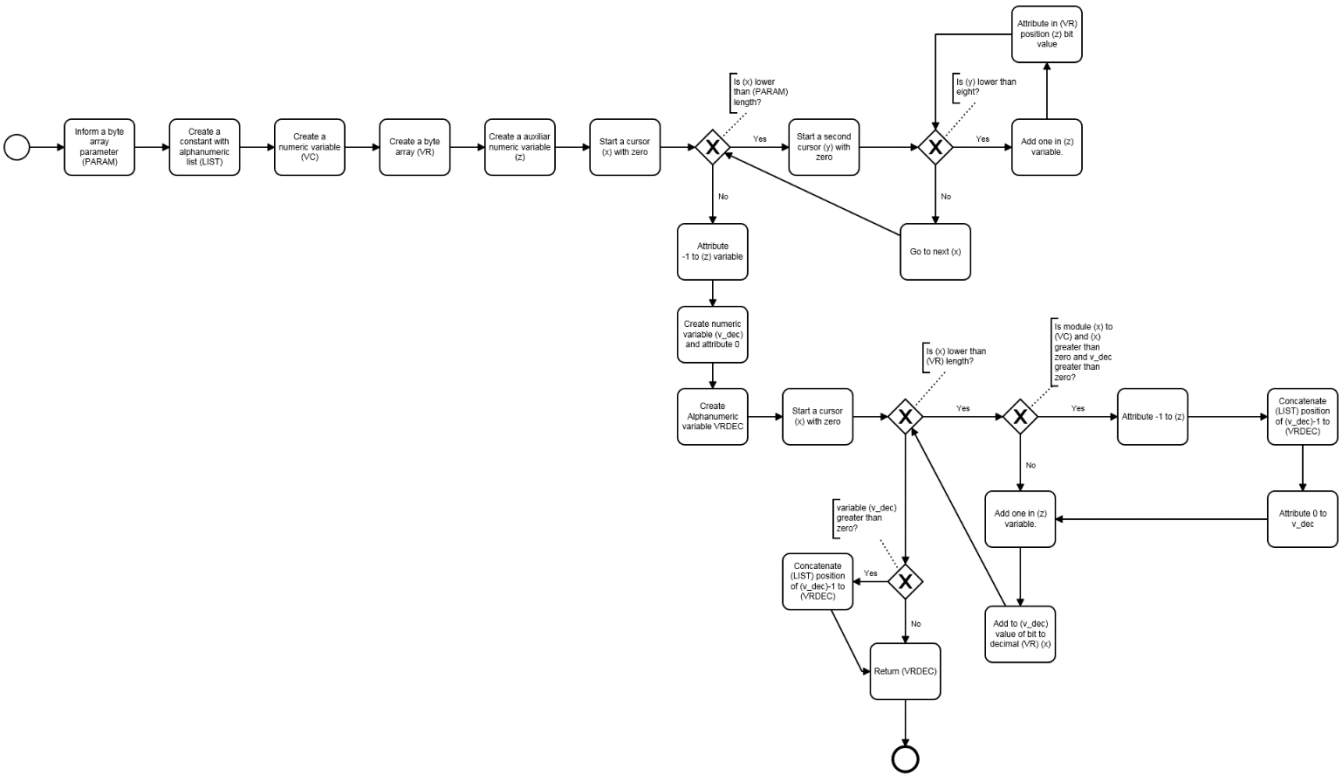
XII. Within the cursor (x) verify that the modulus of position (x) by (VC) is equal to zero, the value of (x) is greater than zero and if the value (v_dec) is greater than zero. If it is, assign to (z) the value of minus one, and after, concatenate in (VR) the character indicated in the position of (v_dec) minus one located in the variable (LIST).

XIII. Still inside the cursor (x) add one more in the value of the variable (z).

XIV. In the cursor (x) assign to the variable (v_dec) the sum of (v_dec) plus the result of the multiplication of VR of the position (x) by the result of the power of 2 raised by (z).

XV. After the termination of the cursor (x), check if there is any value greater than zero in the variable (v_dec), if any, concatenate in (VR) the character indicated in the position of (v_dec) minus one located in the variable (LIST).

XVI. Finally, the variable (VR) will have the contents unpacked.



Registration and use of the system:

The clients who must inform the projects that require the licensing by simply informing the field types and the size for a simulation, and the payment will be calculated based on the amount of transactions made in the blockchain used.

If it is an internal/corporate blockchain, they should indicate the amount of transactions performed periodically, since it is not possible to calculate from the internet.

Main benefits:

- Time optimization;
- Financial economics;
- Energy saving;
- Savings in data traffic;
- Collaboration in decreasing the sizes of databases and blockchains;
- The adaptive system.