Proceedings in Manufacturing Systems, Volume 14, Issue 2, 2019, 55-60 ISSN 2067-9238

NEW METHOD FOR SENDING DATA TO THE CLOUD FROM AN OPC UA CLIENT APPLICATION

Gicu-Călin DEAC[1,*], Crina Narcisa GEORGESCU[2], Cicerone Laurențiu POPA[3], Costel Emil COTEȚ[4], Florina CHISCOP[5]

[1)] PhD Student, Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania
[3)] Associate Prof., PhD, Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania
[4)] Prof., PhD, Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania
[5)] Lecturer, Ph.D., Robots and Manufacturing Systems Department, University "Politehnica" of Bucharest, Romania

Abstract: *The present paper presents a new method for sending data to the cloud from a Client OPC UA Application (Open Platform Communications Unified Architecture), using PNG images as a medium. The data values encoded as pixel colors of images, using different encoding methods for each data type (text, integer, float, Boolean) are also encrypted using an image as a symmetric encryption key and are stored in the cloud in a time base folder structure. This method of sending and storing the data using an encrypted image format assure a better data compression, security and speed, compared with the actual JSON format which is text based and open. Based on the stored image files can be generated some index image files containing the data values of each property (sensor value) for a chunk of time, in this way reducing the total number of images and increase the compression of data, because of the small allocation space on disk and similarity of data that are better compressed by the LZW compression algorithm of PNG image format. This indexed image file can replace the actual SQL or NoSQL databases and can be read by multithreaded agents using multicore processors for faster interrogation and easily replicated to other servers by copying the new generated files.*

Key words: *IIoT, OPC UA, Big Data, Cloud, Data encryption*.

1. INTRODUCTION

Taking into consideration the performance and the effectiveness of a control system, I. Gonzales et al. say that the key factor is represented by the interconnection between sensors, controllers, instruments and cloud services, through a communication network.

Supervising, tracking and automating technological processes for both industrial and non-industrial settings requires efficient transmission of information across communication networks [1]. Wired or wireless channels are used to communicate data. Due to mobility and flexibility, wireless communication has great benefits and will be widely used in IoT (Internet-of-Things). Different wireless protocols such as Wi-Fi, Bluetooth, ZigBee, 3G/4G/5G, RFID, Z-Wave, IPV6 over Low Power Wireless Personal Area Networks (6LoWPAN) and Near Field Communication (NFC) are available [1].

Industrial Internet-of-Things (IIoT) is a term used to refer to the IoT applications in the industrial context and implies the use of sensors and actuators, control systems, M2M (machine-to-machine) communications, cloud storage of Big Data, data analytics and security mechanisms [1].

According to L. Monostori et al. another challenging concept emerges, starting with the Industry 4.0 paradigm: Cyber-Physical Systems (CPS) which means that industrial machines have sophisticated communication and smart capabilities, that devices are linked over the network to detect, monitor and act on physical components in the real world [2]. In order to simplify and optimize this large-scale integration, proprietary protocols and new evolving communication protocols must converge on a prevalent protocol platform [2]. One of the present existing protocols for industrial communication is OPC-UA, a protocol that has been standardized in the IEC 62541 series. OPC UA is an open and secure platform that allows vendor-neutral programmable logic controllers (PLCs) to communicate with each other and up to the manufacturing level and into the production planning or ERP system [7]. This protocol can aid in data analysis, in addition to reducing costs for licensing, staff training, hardware upgrades and system migration. OPC UA is independent of the platform manufacturer and from the programming language in which the applications were developed [7].

At its core, OPC UA defines an asynchronous protocol (built on TCP, HTTP or SOAP) that defines the exchange of messages on raw connections via sessions, alongside secure communication channels [8]. This protocol replaces the OPC Classic protocol, retaining all the functionality of its predecessor but being more portable. Because OPC Classic was built upon a Microsoft communication technology – the Distributed Component Object Model (DCOM), this protocol was bound to Windows, which became increasingly limiting.

[*] Corresponding author: Gicu-Călin Deac, University "Politehnica" of Bucharest, Splaiul Independenței 313, Bucharest 060042
Tel.: 0040 741073007
E-mail address: *george.deac@impromedia.ro* (Gicu-Călin Deac)

On the OPC Foundation website at https:/opcfoundation.org, the standard itself can be purchased from IEC or downloaded for free. Open 62541 deploys the OPC UA binary protocol stack as well as a Software Development Kit (SDK) for client and server.

The Micro Embedded Device Server Profile is presently supported along with some extra characteristics [8]. The OPC UA protocol describes 25 built-in types of data and three methods to combine them into higher-order kinds: arrays, constructions, and unions. Only the built-in data types are manually described in open 62541. All other kinds of information are produced using normal XML definitions [7].

The OPC UA Built-in Data Types are: Boolean, Byte, SByte, Int 16, UInt 16, Int 32, Int 64, UInt 64, Float, Double, String, Date Time, Status Code, Guid, Xml Element, Node Id, Expanded Node Id, Byte String, Qualified Name, Localized Text, Numeric Range, Extension Object, Data Value [8].

The OPC UA architecture can be represented as in Fig. 1 [6], where Data producing Devices are PLC (programable Logic Controllers), DCS (Distributed Control Systems), IED (Intelligent Electronic Devices), PAC (Programmable Automation Controllers) or DDC (Direct Digital Control) and Data Consuming Applications could be: HMI / SCADA, Historians, databases, MES (Manufacturing Execution Systems) or IoT & Big Data Platforms.

The OPC UA server is like a Middleman and can discover all the nodes in the network, can map them and get all the values from the sensors and processes, based on a request from OPC UA Client applications.

The OPC UA standard was adopted already by many industries, for example, Arburg GMBH and Co KG in machine manufacturing, AREVA in the field of renewable energy, for remoted wind farms, KU Leuven for ground-based observatory control and Elster GMBH in discrete manufacturing to connect shop floor to SAP ME top floor [7].

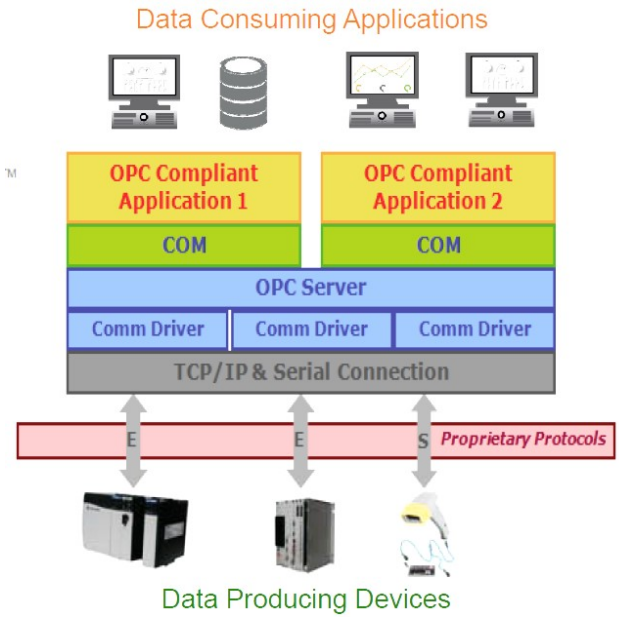

Fig. 1. The OPC UA architecture.

2. IMAGE BASED DATA STORAGE METHOD

In the case of an IIoT platform, the data from sensors and processes must be uploaded to the cloud and stored in time series databases to be available for different applications like predictive maintenance, process optimization, and reports. The uploading process and the storage imply a strong security policy to be implemented, to prevent unauthorized data access [4]. The quantity of data uploaded being also very high, some compression algorithms must be taken into consideration.

The present OPC UA clients and proxy are using JSON files to upload data to the cloud, this kind of files being easy to read in the case of a man-in-the-middle type of cyber-attack.

The present paper proposes a Client OPC UA application that will read all the values from the OPC UA Server, on timed cycles with a predefined frequency (each second for example) and use a novel method for encoding and encrypting of data and automatic uploading in the cloud.

The method consists of creating full-color images (16 million colors) whose pixels are generated based on numerical or alphanumeric values to be stored.

The color of each pixel can be defined by the three-component R, G, and B with values between 0 and 255, respectively 0 and FF in hexadecimal. The image size can be defined according to the number of values to be archived and their type. For example, to store 10000 positive integer values, a 100×100 pixel image will be defined.

For encoding, there are several methods, depending on the type of values to be stored:

a. In the case of positive integers between 0 and 16777215, the encoding can be done directly by converting the decimal number to hexadecimal (ex. 16777215 becomes FFFFFF), the six-digit hexadecimal string explodes into three groups of two digits and each of this group is (R = 255, G = 255 and B = 255, in this case, a white pixel corresponding to the maximum value that can be stored);

b. In the case of positive and negative integers, the same encoding variant as above may be used by having the maximum value to +/- 8388607, the numbers being encoded by summing the storage value by 8388607 and decoding being made by subtracting this value;

c. For float numbers they can be converted to 16-digit hexadecimal values: 00.00.00.00.00.00.00.00, requiring three pixels for storage (R1G1B1, R2G2B2, and R3G3, the last component B3, is not used). Thus, any float value with any number of decimals, positive or negative, (in IEEE 754 double-precision standard, the full range of numbers is -9.999999×10^{96} through 9.999999×10^{96} [9]) can be archived using a group of three consecutive pixels. The image will have the number of pixels equals three times the number of archived float values;

d. For String, alphanumeric values (UTF8), the conversion of each alphanumeric character into hexadecimal is made according to the UTF8 encoding table or a custom defined table defined (for example A

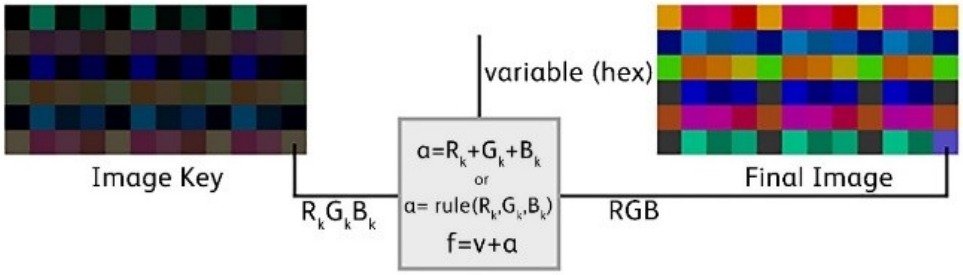

Fig. 2. Image generation.

becomes hex: 41, *z* becomes hex: 7a), so each pixel can store a hexadecimal to decimal number of 3 alphanumeric characters, one for each component R, G, and B. The image will have the number of pixels equal to one third of the total number of characters.

e. Booleans values can be stored as alphanumeric values 0 and 1.

For encrypting data of any kind, corresponding to the above-mentioned encoding methods, an encryption image with the number of pixels equal to the number of stored variables is used.

The automated generation of the image key is based on image size and using random values for Rk, Gk, Bk. When variables (properties values) are encoded at each variable converted in hexadecimal will be added the hexadecimal value returned from the correspondent pixel in the image key (for the first variable will be used the first pixel, for the second variable the second pixel and so on, or we can scramble the pixel order based on an established rule). Can be used the hex sum or RGB values or only one hex value (from R, G, or B based on a variable ID established rule) (Fig. 2).

If the resulting value of R, G or B is greater than 255, (FF in hex) the result will be decreased by 255 (FF in hex). The decryption and decoding process is analogous but in reverse. In other words, from the hexadecimal value returned by decoding the RGB values of each group of pixels (based on variable type and encoding method) we will subtract the hexadecimal value returned by RGB of the correspondent pixel in the Image Key based on the established rule and then based on the encoding method we will decode the variable value. If the values obtained are negative, this value will be increased with 255 (FF in hex) before decoding the variable value.

In a real-life scenario, the method can be used for encoding and encrypt mixed data types. In this case, can be included some descriptive pixels for the device ID, timestamp, number of floats, number of varchar variables and their size, number of Booleans. (for example, first three pixels for the device ID, next three pixels for the timestamp in milliseconds, next pixel for the number of floats, next pixel for the number of varchar variables, next pixel for the size of varchar variable: ex: 24 characters, next pixel for the number of Booleans).

In IoT applications, the OPC UA Client application will generate the images based on the device properties values, with a predefined frequency (example: 1 time/sec) and send them to the cloud using a socket, or any other protocol. The PNG images names can be constructed on the following model:

deviceUniqueID_timestampInMiliseconds.png
example: kep35_31494270124240.png

and will be stored on the cloud in an arborescent file structure based on a model like:
/deviceUniqueID/t1/t2/t3/t4/t5/t6/t7/deviceUniqueID_timestampInMiliseconds.png
example:
/kep35/1/4/9/4/2/7/0/ kep35_1494270124240.png

The folder structure can be generated based on timestamp by exploding the first 7 characters and store the images inside the last subfolder (*t7*). This storing model allows a good management of time-series data, to read, copy and replicate only the desired intervals. This model also allows real-time interrogations of device state at a specific time.

The PNG nondestructive compression, based on LZW offers a great compression rate, comparing to JSON files, the same amount of data being much smaller.

For this research was developed an IoT application (Fig. 3) based on the presented method and were generated a sample of data to perform comparative tests [3]. The application includes also a graph view of time series data and some machine learning algorithms like *k*-means and linear regressions.

Was simulated a production process that has 100 different tags (sensor values) and using an OPC UA client application the data was sent to the cloud databases in both formats: JSON and PNG.

For a proper comparison was used an interval of 1 million records for each tag, with a frequency of 2 sets of data per second, having in this case:

Historical data between 08 05 2018 18:57:01 and 14 05 2018 13:50:20 (1 million records of 100 tags 0.5 sec rate).

Properties values: 59 floats, 24 texts (18 characters length), 17 Booleans Raw data sent to the cloud: 1 million JSON files each contain 100 properties values: transfer to the cloud approx. 16 GB, 1 million PNG images each contain 100 properties values: transfer to the cloud approx. 1 GB.

The amount of data transferred to the cloud in the case of PNG format is significantly smaller (16 times smaller) comparing to the JSON format, for the same amount of data values. This means that the PNG format is more suitable for packaging the data for transport to the cloud.

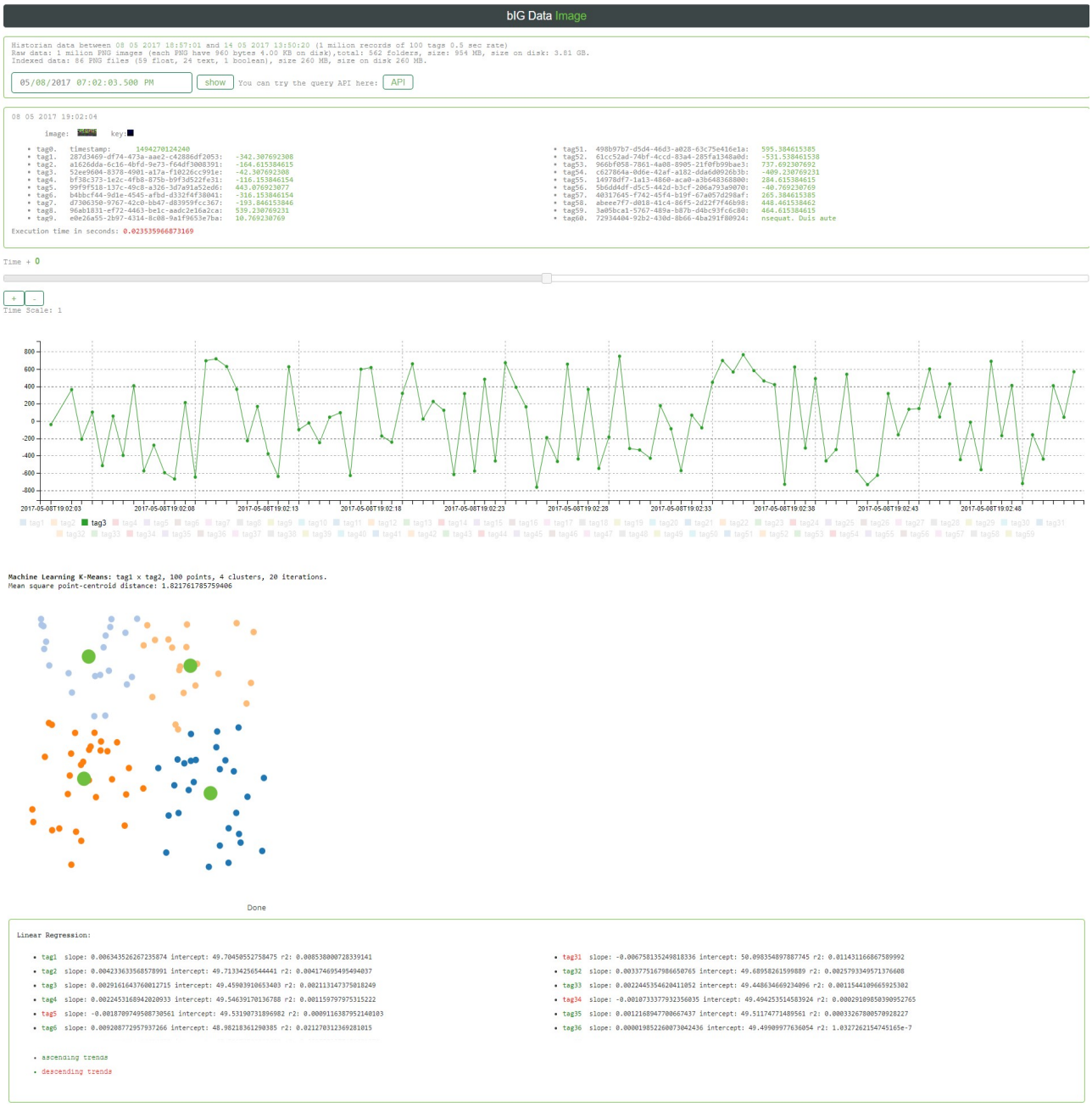

Fig. 3. IIoT web application screen.

Also, the PNG is encrypted using a symmetric image encryption key and in the case of a man-in-the-middle attack provides strong protection, comparing to JSON files, that are protected only by the SSL encryption of the OPC UA protocol. For storing data in the cloud, a Cassandra Database Node and an image storage model using PNG files were used.

Stored data: Cassandra Database Node, containing the JSON files, needs 18.5 GB on the storage. In the case of data retrieval, average time reading of properties from Cassandra using a SELECT WHERE statement took an average of 0.185 sec. Raw data snapshots in PNG format: Storage on cloud: total: 562 folders each PNG have 960 bytes, 4.00 KB on disk (the stored size is bigger because of the allocation units on the disk). There are in total: 562 folders, 1 million PNG with 960 bytes size: 954 MB, size on disk: 3.81 GB. Average time reading of properties from Image Database using SELECT WHERE clause was 0.0026 sec.

The PNG model is much faster than the Cassandra database (about 70 times faster), the speed difference is even more radical in the case of a large database.

The advantage of the proposed method relies on the fact that PNG is stored in a time-based folder architecture and the speed of retrieval of the information based on a timestamp does not depend on the size of the database, because is not needed to load the entire database in memory to execute the selection query. The replication of the database is also much simpler in the proposed method and consist of simply copying the folder structure with its content to a new machine. Can be also selected only a needed period for data, by copying only the specific folders.

The data retrieval mechanism in the proposed method needs significant smaller hardware resources, compared to other databases such as Cassandra that needs to load the data in memory to perform the queries.

3. CONCLUSIONS

Using the proposed method, one can achieve a secure, fast and efficient way to send and store data in the cloud.

The data in the case of the proposed method is encrypted using symmetric encryption, each value being differently encrypted based on the correspondent pixel color on the image key.

Because of the LZW compression, the PNG is much smaller than JSON files and require 16 times less bandwidth to send the same amount of data, or in the case of the same connection bandwidth, the data sending is about 16 times faster.

In the cloud, using this model of time-based folder storage, the disk space is about 6 times smaller comparing to classical databases.

The data retrieval in the case of the proposed model is much faster (more than 70 times faster) than Cassandra Node and require much lower hardware resources, because the retrieval of data can be done without the loading of entire database (or part of it) in memory, but directly accessing the images from the correspondent folder and reading the values of the correspondent pixels.

This method can be further developed in order to use this image database as an alternative to actual SQL or NoSQL existing databases by creating based on the Raw image PNG files, some indexed images for each property (sensor value) for a chunk of time (one hour, one day etc., depending on the data acquisition frequency).

For example, choosing a day (24 hours) in this example with 100 different properties (distinct sensors data) will be generated at the end of a day, based on a CRON script 100 different indexed images, containing each one the values of a specific property in this 24-hour interval.

To generate this image, RAW images will be read based on the time from 0: 00: 00: 00 to 23: 59: 59: 50 (having data stored 2 times per second).

The corresponding values in the RAW image will be decrypted and decoded, then the indexed image files will be created.

The indexed image file can be also encrypted using an image key and can have also descriptive pixels for location, device, sensor ID, timestamp, etc. In this way, for one day of operation, there will be 100 separate indexed files, each containing 172800 values, instead of having 172800 RAW images with 100 distinct property values. These images will be stored in a folder structure such as:

/deviceUniqueID/year-month-day/propertyID/propertyID_year-month-day.png

Using this indexed image, the space needed for storage will be even smaller and the reading speed of data for reports over a long period will also be much smaller because of a small number of files with bigger sizes and the allocation unit on the disk does not count anymore. Also, because the values of a distinct sensor could be constant on a small period, the LZW compression of the PNG image file will compress better the image.

Even more, the files can be read by multithreaded agents using multicore processors or CUDA for faster interrogation. Also, the image files can be read by many distinct processes at the same time, in parallel not by using a queue like in the classical databases. This method of storing big data using images could replace classical databases like PostgreSQL or Cassandra.

An IoT implementation of the method can be seen on Fig. 3 and accessed on [3].

On the same link, one can see a pseudo SQL API by clicking on API button (Fig.4). A property based on id (integers from 1 to 100) can be selected. The search will return a web address like:

http://vps.impromedia.ro/dataimage/query.php?id=66&s=2017-05-08T18%3A57%3A01&o=100&c=&n=&d=&m=&k=&r=&l=&key=0

The query link can be accessed from any external application sending the proper variables:
id – the ID of a property (sensor);
s – the starting timestamp of the query
(in a date time format ex: 2019-05-08T18:57:01:00);
o – offset (number of timestamp base increments);
c – first condition of the WHERE clause
(can have this value: <, <=, =, >, >=);
n – the first value of the condition;
d – the second condition of the WHERE clause
(can have this value: <, <=, =, >, >=);
m – the second value of the condition;
k – ordering by variable
(can have this value: t – for time base order and
v – for value order);
r – ordering type:
(can have the value: ASC and DESC);
l – limit (numeric value);
k – key (numeric value 0, 1, 2, … n you can specify a different image encryption key);

For example, in order to select for the sensor ID = 10 first 50 values starting from 08.05.2019 18:57:01:00 into an interval of 10000 records, WHERE sensor value > 10 and sensor value <= 100, ordered by value ASC can be created a web query like:

http://vps.impromedia.ro/dataimage/query.php?id=10&s=2017-05-08T18%3A57%3A01&o=10000&c=>&n=10&d=<=&m=100&k=v&r=ASC&l=50&key=0

In this research was also created an experiment to generate random numbers, to insert text using the encode/decode process. The experiment application can be seen on [10]. In this application, when you generate an image based on the randomly generated numbers and display this image, by hovering the mouse cursor over this image pixels, you can see the property ID and the value stored in each pixel.

As future research, the compression of PNG files will be further optimized, using a Median Cut quantization and Voronoi iteration (K means).

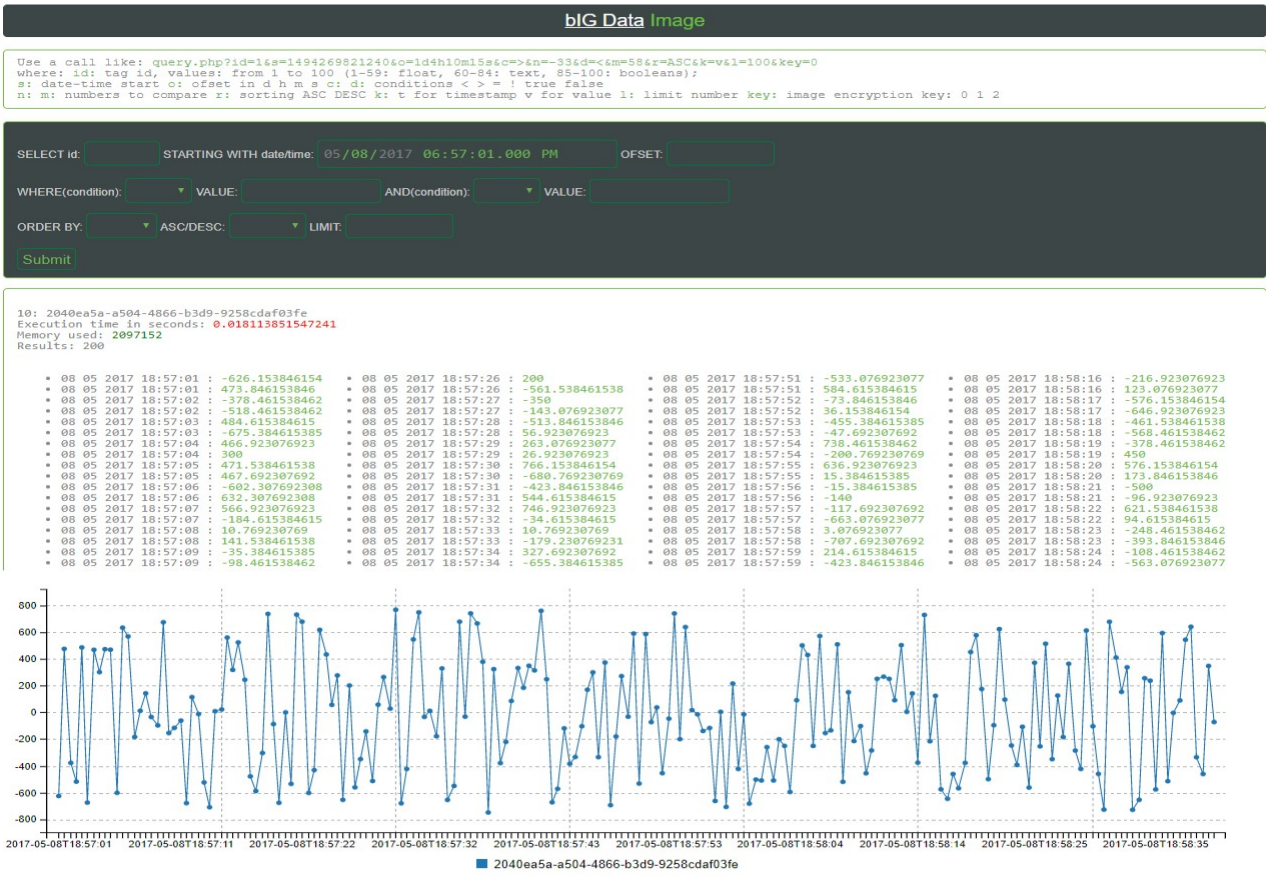

Fig. 4. Pseudo-SQL API interface.

Also, will be created using FFmpeg lossless video files from PNG images to store the telemetry data.

The framerate of the video will correspond to data retrieval interval (eg. 2 frames/sec) and experimenting with different lossless video compression codecs will be choose the better one.

Using the video archiving format, all the data will be stored in a single file for an hour or even a day. To read the stored data from the video file, it is possible to read the correspondent frame, based on the timestamp.

A backup application based on the proposed method can be created and will be used for backing up the data from any database, in order to be archived or transferred. The application will generate images based on the database content and will export SQL format files from images, to restore the database data.

The graphical representation mode of the data can be used for monitoring real-time data. For example, to compare the real-time streamed sensor values with some reference values generated from a digital process twin, can be displayed a resulting image created on the fly by inverting the reference image and overlay it with the streamed image. In this way, if the reference values are equal with the real one, the resulting image will be entirely black. When some difference will appear, some pixels will be lighted. By hovering the mouse on those lighted pixels, can be seen in real-time the difference from the reference values.

REFERENCES

[1] I. Gonzales, A. J. Calderon, A. J. Barragan and J. M. Andujar, *Integration of Sensors, Controllers and Instruments Using a Novel OPC Architecture* Published online 2017.

[2] L. Monostori, B. Kadar, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, K. Ueda, *Cyber-physical systems in manufacturing.* CIRP Ann-Manuf. Technol. 2016.

[3] *Cloud application based on the new proposed method and data sent by the OPC UA Client application*, available at: `http://vps.impromedia.ro/dataimage/index.html`

[4] Airehrour D., Gutierrez J., Ray S.K., *Secure routing for internet of things: A survey.* J. Netw. Comput. Appl. 2016

[5] Deac G.C. – Patent application no. A 2017 00174 entitled *Method and Online Encryption System, Transmitting Storage and Reading of Large Data Volumes* (OSIM).

[6] The Center of OPC Solutions: `http://opchub.com`.

[7] Podnar Žarko I., Broering A., Soursos S., Serrano M. *Interoperability and Open-Source Solutions for the Internet of Things.* InterOSS-IoT 2016. Lecture Notes in Computer Science, vol 10218. Springer, Cham, 2017.

[8] Open 62541 Docs: `https://open62541.org/doc/current/`

[9] IEE754: `https://en.wikipedia.org/wiki/IEEE_754`.

[10] *Encoding/Decoding experimental application*: `https://transmissiongate.com/dataimage/index.html`.