BIG DATA AGGREGATION ALGORITHM FOR STORING OBSOLETE DATA

Assoc. Prof. Alieksieiev V. PhD.[1,2], Ivasyk G. PhD.[1], Assoc. Prof. Pabyrivskyi V. PhD.[1], Assoc. Prof. Pabyrivska N. PhD.[1]
[1] Lviv Polytechnic National University, Lviv, Ukraine

[2] Vladyslav.I.Alieksieiev@lpnu.ua, https://orcid.org/0000-0003-0712-0120

Abstract: *Many contemporary IoT systems do produce a large scale of data. While a new portions of data come to data storage (database etc.) all the previously stored data become obsolete. Most of such obsolete data become excessive and can be needed only to see general trends or anomalies. This research offers an algorithm of data aggregation to minimize the amount of stored obsolete data according to defined business rules. Some modifications of algorithm are discussed to fit different kind of business requirements. There is also a comparison of two methods of data merge in algorithm, quantization and clustering, was made.*

Keywords: BIG DATA, OBSOLETE DATA, DATA SERIES, TIME SERIES, IoT, AGGREGATION ALGORITHM

1. Introduction

Let's start from considering the same IoT based monitoring system as it was described earlier [1]. Typically, we have a large number of devices each with a set of sensors. The data from all the devices and all the sensors is gathered on a server (a cloud storage or a database at least). Using this data for online monitoring system causes previous data to become obsolete when several portions of new data comes to server. It is clear, for example, that data prior to current day, week or month can be defined as an old data according to business specifics.

One can ask if there is any need to store such data at all. The answer would be "yes, there is", if we expect to analyze collected data and gain some new knowledge or make conclusions about environment or system we are monitoring. Many researchers and practitioners in area of data analysis declare statements similar to [2, p.2]: "data accumulation can enable deeper insights and help us to gain more experience and wisdom". There are many evidences of great performance of time series analysis already [3, p.5] and there is a number of solutions for time series databases [3, p.11]. Therefore, the answer seems to be obvious – we need that data.

This means for the purpose of data analysis, on one hand, we have to store many data gathered from all devices and sensors of our monitoring system. However, on the other hand, not completely all particular values of the data in the data storage are necessary. Moreover, some data can even bring excessive information that can be discarded easily. Each researcher or data analyst or data engineer will decide by himself about the amount of data needed to make some reliable conclusions. Nevertheless, fast growing volumes in data storage can make it impossible or unprofitable to store all original data for previous years, for example. This situation requires some balanced judgement with respect to reasonable and effective software solution.

2. Prerequisites and means for solving the problem

First, we must define the rule to separate obsolete data and current data. Second, we should determine types of data values stored and considered to become obsolete. Then, finally, third, we can build algorithms for obsolete data aggregation.

Considering data can become obsolete one may think of necessity to use such data again. In case of real-time monitoring system, current data is data needed to display the latest state of a system. Turning back to early-discussed system [1] we can assume N_{window} as a number of current values and consider these values necessary to display the latest state. If the time period of a new data package comes to server is t_p, then the whole period of $t_p \times N_{window}$ has current data. If the monitoring system must display both the latest state and some previous dynamic, then the current data can be defined as $t_{current}=(N_{display}+N_{window})\times t_p$, where the $N_{display}$ is the number of values to be displayed and $(N_{display}+N_{window})$ stands for number of packages currently needed. However, the data prior to $t_{current}$ period yet cannot be considered as obsolete if business requires seeing details of some historical period. This historical period can be a single day or a week, or a month or even a year. Nevertheless, we should be exact in definition of obsolete data, so we should rely on accuracy of monitoring and probability of necessity of data. In general, we can understand necessity of data as repeated requests for that data and when there are no more requests means the data is obsolete. This can be compared to the probability of device becomes broken in theory of reliability or the species will survive during some period [4; 5]. That is why we use exponential distribution for the probability the data will be no longer in use. One can use PDF in the form:

$$f(x,\beta)=\begin{cases}0, & x<0,\\ \frac{1}{\beta}e^{-\frac{x}{\beta}}, & x\geq 0.\end{cases}$$

Here, x stands for the random variable meaning duration of time to "survive" and β is a "survival parameter" (probability to survive during time x). Next, we can define $t_{obsolete}=t_{recent}+t_{current}$, where t_{recent} is the time, before current time and needed, for example, for purpose of operational analysis. Having a number of values N_{recent} we can define $t_{recent}=t_p\times N_{recent}$. So, $t_{obsolete}=(N_{recent}+N_{display}+N_{window})\times t_p$. One may consider probability of data necessity:

$$\int_0^{t_{obsolete}} f(x,\beta)dx=\int_0^{t_{obsolete}}\left(\frac{1}{\beta}e^{-\frac{x}{\beta}}\right)dx=1-e^{-\frac{t_{obsolete}}{\beta}}\geq 1-\varepsilon.$$

This is the CDF at $t_{obsolete}$, naturally. The value obtained we compare to business requirements about historical data involvement. At last, we have the expression to bind $t_{obsolete}$ and β:

$$t_{obsolete}\geq \beta\cdot\ln\varepsilon^{-1}. \quad (1)$$

Value $\varepsilon\in(0,1]$ can be set to 0.1, 0.05, 0.01, 0.005 or 0.001 according to business requirements for historical data involvement for purpose of analysis of operations (see Table 1).

Table 1*: Business requirements for historical data involvement.*

ε	$\ln(\varepsilon^{-1})$	Business requirement description
$1/e\approx 0.368$	1	Light involvement
0,1	≈2,3	Small involvement
0,05	≈3	Medium involvement
0,01	≈4,6	Moderate involvement
0,005	≈5,3	Strong involvement
0,001	≈6,9	Hard involvement

One can solve both direct and inverse problems using relation (1) between $t_{obsolete}$ and β:

- Direct problem (finding $t_{obsolete}$) – when the data become obsolete according to expected data necessity time β?
- Inverse problem (finding β) – what is the expected data necessity time according to time $t_{obsolete}$ the data has become obsolete?

Key parameters for time definitions in our monitoring system are t_p (the period between data packages come to server) and N_{window}, $N_{display}$ and N_{recent} (a few different values of number of packages). Through the definition of $t_{obsolete}$ we can find yet

unknown value for N_{recent} or, vice versa, based on operational analysis needs given by N_{recent} we can find the expected data necessity time.

Any of three parameters (N_{recent}, $t_{obsolete}$, β) can be used initially to fit business requirements with respect to value of ε. In case of our system we used ε=0,05 and β=31 days=744 hours (about 1 month). This yields $t_{obsolete}$=93 days=2 322 hours (about 3 months) and N_{recent}=8 928–48–8=8 872, while t_p=15 minutes=1/4 hour and N_{window}=8 and $N_{display}$=48 to display last 12 hours at once.

Now, as we defined the obsolete data criteria, we can move forward to build an algorithm for data aggregation.

3. Solution of the examined problem

Incoming data packages can deliver some kinds of values according to type of sensor and type of gathering.

Sensors can be one of three types:

- Alert – signals of normal and alert states (movement detection, smoke detection, some chemicals detection, etc.), collected as a Boolean value (true or false).
- Value – quantitative values gathered as sensor values of some type (temperature, illumination, voltage, loudness, etc.).
- Normalized – same as value but compared to some "normal" value or predefined levels of value. Normalized value goes to the server as a level number (0, 1, 2, 3 …) and in case of only two levels is similar to alert.

Types of gathering correspond to period characteristics:

- Immediate – gathered and sent to the server immediately.
- Summarized – gathered and accumulated within a sum (count, minimum, maximum, average, mode, etc.) during predefined period. One may consider very different types of values depending on aggregate function used to accumulate data.

See Table 2 for possible combinations of different kinds of values coming from sensors. Note, that these kinds of values in Table 2 are the values coming from device and we are to discuss further aggregation for obsolete data, but not at the level of a single device.

***Table 2**: Types of incoming data.*

	Immediate	**Summarized**
Alert	Simple alert indicating appearance of some alert state (detection alerts)	Counters (count of door openings, number of visitor passes, etc.) used during predefined period
Value	Simple value sent to the server – the case of a very detailed monitoring	Value aggregate (like sum, min/max, average) used to detect some state or going beyond some limits
Normalized	Detects shift or switch between some states	Level aggregate (like count, min/max, mode) used to detect some state or shift (switch) between states

Now we can discuss the most appropriate types of obsolete data for application of aggregation algorithm. Immediate alerts can be aggregated the same as summarized alerts with a sum or a counter. Immediate or summarized values in the simplest way can be aggregated with some aggregate function (sum, average, etc.). However, there can be also used a quantization or clustering to merge obsolete data. If we have some values split by some natural levels, then quantization is possible. If there are no any predefined levels for values, but values reveal some grouping, then clustering should be in use. Normalized data is naturally good for quantization method of aggregation. Nevertheless, using some simple aggregate function can be a good way to aggregate immediate normalized values.

3.1. Quantization

The key idea of this method for data aggregation is to make averages within predefined quants. This means to define (according to business rules or nature of data) some levels of possible values (quants) and then to split timeline (because we have a time series) into intervals, within which the average values will be found.

Short description of an algorithm steps:

- Form quants (levels) of values according to minimum and maximum values of data sample and minimum and maximum boundaries for normal (appropriate) values. At least three quants (levels) we shall have. Large quants (those having outlying boundaries) can be split into additional levels, and finally K the number of quants is found.
- Split timeline into intervals according to previously formed quants. This means we have the same interval while the current value remains within quant of previous value. When only the value goes outside the previous quant, a new interval starts.
- Calculate average values within each interval.
- Replace old set of values with averages (with respect to start point of each interval).

Now we can make some formal definitions for the algorithm.

Let $\boldsymbol{X}(\boldsymbol{T})$ to be the set of incoming data depending on time ($\boldsymbol{T}$ – timeline set). We can have X_0 as a "normal" value and Δx as a appropriate deviation for x. This means we have appropriate maximum and minimum boundary values $X_{0,max}=X_0+\Delta x$ and $X_{0,min}=X_0-\Delta x$. However, if we have business rules saying $X_0 \neq (X_{0,max}+X_{0,min})/2$, then we can use actual values of $X_{0,max}$ and $X_{0,min}$ instead. Next goes the algorithm:

1. *Quantization.*

1.1. Find maximum and minimum values in the set and boundaries:

$$x_{max} = \max(\boldsymbol{X}(\boldsymbol{T}))$$
$$x_{min} = \min(\boldsymbol{X}(\boldsymbol{T}))$$
$$x_{1,max} = \max(x_{max}, X_{0,max})$$
$$x_{1,min} = \min(x_{max}, X_{0,max})$$
$$x_{-1,max} = \max(x_{min}, X_{0,min})$$
$$x_{-1,min} = \min(x_{min}, X_{0,min})$$

1.2. Three evident quants in the set of quants $\boldsymbol{Q} = \{ (x_i, x_{i+1}] \}_{i=1\ldots K}$ with K=3 now are:

$$[x_{-1,min}, x_{-1,max}], (x_{-1,max}, x_{1,min}], (x_{1,min}, x_{1,max}]$$

1.3. Find minimum and maximum distance between values:

$$d_{min} = \min (|X_{0,max}-X_0|, |X_0-X_{0,min}|, |x_{max}-X_{0,max}|, |X_{0,min}-x_{min}|, |x_{max}-X_0|, |X_0-x_{min}|), \quad (2)$$

$$d_{max} = \max (|x_{max}-x_{min}|, |X_{0,max}-X_{0,min}|, |x_{max}-X_{0,min}|, |X_{0,max}-x_{min}|). \quad (3)$$

1.4. Find some additional quants to split very wide quants. One may use rule $K = ceil(d_{max}/d_{min})$, where *ceil* function means nearest integer not less than argument. But such rule may cause K to be unreasonably large, if there is one very narrow quant. As a criterion to split some quant we offer to compare width of each quant with the average width and split those quants having width twice bigger than average. If we have quant widths $d_i=|x_{i+1}-x_i|$ and the average width $d_{avg}=(\Sigma_i d_i)/K=(\Sigma_i |x_{i+1}-x_i|)/K$, then split procedure should follow the formal rule for each quant from $\boldsymbol{Q}$:

$$\text{if } d_i > 2 \cdot d_{avg} \text{ then}$$
$$K := K+1,$$
$$x_{i+1/2} = (x_{i+1}-x_i)/2,$$
$$\boldsymbol{Q} := \{ (x_i, x_{i+1/2}], (x_{i+1/2}, x_{i+1}] \} \cup \boldsymbol{Q} \setminus \{ (x_i, x_{i+1}] \}.$$

1.5. To avoid too much quantization one may suppose $3 \le K \le 7$.

1.6. Quantization result is a the set of quants $\boldsymbol{Q} = \{ (x_i, x_{i+1}] \}_{i=1\ldots K}$.

2. *Splitting timeline intervals.*

2.1. Start finding set of timeline intervals $\boldsymbol{I} = \{ (t_{n-1}, t_n) \}_{n=1\ldots N}$ with t_0 as a start point and t_N as a final point and N as a total number of time intervals. The rule to find inner split points:

$$t_n = \max\{\, t \in \boldsymbol{T} \mid \forall t > t_{n-1}\ \exists! i \neq j : x(t) \in (x_i, x_{i+1}],\ x(t_{n-1}) \in (x_j, x_{j+1}] \,\}$$

2.2. Splitting timeline intervals result is a set of intervals $\boldsymbol{I} = \{ (t_{n-1}, t_n) \}_{n=1\ldots N}$.

3. *Calculating averages.*

3.1. Find average value x_n among N_n values of $x(t)$ for each interval (t_{n-1}, t_n):

$$x_n = \frac{1}{N_n} \sum_{x(t_{n-1}) < x \le x(t_n)} x.$$

3.2. Calculating averages result is a new data set of averages $\boldsymbol{X}_{avg}(\boldsymbol{I}) = \{ x_n(t_n) \}_{n=1\ldots N}$.

The algorithm described yields to make a new set of average values $x_n(t_n)$ where $n \in [1, N]$ and resulting number of points N is expected to be much less then initial one.

3.2. Clustering

Data clustering is a well-known technique and nowadays has many applications, including data compression [6, p.283]. Discussing clustering for our data we must mention, that we have a time series and only one dimension (sensor value, but not a time) fits conditions for clustering. This means that the result of the clustering should yield, the same as quantization, new data set of averages $\boldsymbol{X}_{avg}(\boldsymbol{I}) = \{ x_n(t_n) \}_{n=1\ldots N}$. Unlike to quantization we shall not have predefined levels of values, so we can also call this method "adaptive quantization".

Now, we call the cluster a closely located sequence of values. Once again mention, that this closeness should be considered via the value and a sequence via a time. That is why we can do get nearly the same result as in case of quantization. Initially we can consider the number of clusters equal to the number of values in the data set. We can use then a kind of agglomerative clustering algorithm similar to those in [7], with a Chebyshev distance formula [8]. With respect to only single dimension, this would be one of the easiest distance formulas:

$$\rho(x_{i-1}, x_i) = |x_i - x_{i-1}|.$$

Calculating minimum distances between nearest neighbors from each two adjacent clusters (with K as the current number of clusters) then looks like

$$\min \{ \rho(x_{i-1}, x_i) \}_{i=1\ldots K}.$$

Repetitive merge of clusters requires a criterion to stop the procedure and this criterion could be

$$\min \{ \rho(x_{i-1}, x_i) \}_{i=1\ldots K} \ge d_{min}.$$

The d_{min} here can be the same as in (2). However, according to algorithm flow one can choose any appropriate stop point to match some particular value of d_{min}.

4. Results and discussion

Analysis of quantization algorithm reveals some its weaknesses.

First, there is a great loss of accuracy. The algorithm is good in showing general qualitative plot of keeping or breaking some normal conditions, but it loses absolutely the quantitative plot. The algorithm aims to show periods of bad values (values outside normal boundaries), but not the kind of these values hidden by averages. This looks both strong and weak sides of the algorithm.

Second, there is a case of less applicability when values do often "jumps" between quants. For example, voltage may have an oscillatory behavior. One may consider normal voltage as 220 V±15V, then the voltage close to upper (234V) or lower (206V) boundary may reveal often jumps over that boundary. Such behavior may yield very poor compression or even no compression due to short period of value stay within each quant.

Clustering also has some great faults. The key problem can be that clusters close to some boundaries may, on one hand, hide abnormal values behind averaged values or, on the other hand, show larger abnormal area, than it really was. This means that clustering allows to keep better accuracy, compared to quantization, meanwhile it does not fit the requirement to keep showing boundaries breaches.

Thus, making decision about the algorithm to use (quantization or clustering) requires understanding the nature of business requirement about viewing the obsolete data. For example, if the key value is to keep accuracy close to initial data, then clustering may fit. But if the key value is to keep abnormal values or boundaries breaches, then quantization should pass. Figure 1 shows the graphical example of quantization with respect to requirements about boundaries of normal values.

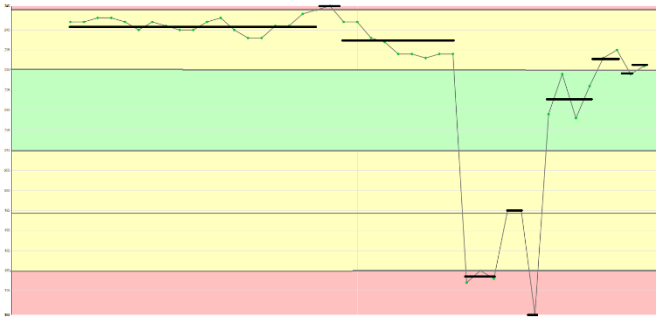

***Fig. 1** Example of quantization for voltage data aggregation*

5. Conclusion

The algorithm of obsolete data aggregation have been discussed. Here we offer the criterion for considering data as obsolete and its connection to particular monitoring system. Using formula (1) allows us to fit different kinds of business requirements about describing data as obsolete. According to different types of obsolete data there were prescriptions made to use different kinds of aggregation. Also there was a detailed algorithm presented for quantization with respect to particular business area and time series specifics. As an alternate solution for quantization, a general approach of clustering algorithm was also offered.

6. Literature

1. Alieksieiev V. About the problem of data losses in real-time IoT based monitoring systems / V. Alieksieiev, O. Gaiduchok // Mathematical Modeling – Scientific-Technical Union of Mechanical Engineering "Industry 4.0" (Sofia, BULGARIA), 2017 – Year I, Issue 3/2017. – P.121–122.
2. Yifei Lin and Wenfeng Xiao. Implementing a Smart Data Platform: How Enterprises Survive in the Era of Smart Data — O'Reily, 2017. – 68 p.
3. Ted Dunning and Ellen Friedman. Time Series Databases: New Ways to Store and Access Data — O'Reily, 2015. – 80 p.
4. Exponential distribution // Wikipedia.org. – Jan. 31, 2018. – https://en.wikipedia.org/wiki/Exponential_distribution
5. Allen B. Downey. Think Stats. — O'Reily, 2015. – 225 p.
6. Jain A., Murty M., Flynn P. Data Clustering: A Review. // ACM Computing Surveys, 1999. – Vol. 31, No. 3. – P.264-323.
7. Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. // ArXiv.org, 2011. – https://arxiv.org/pdf/1109.2378.pdf
8. Chebyshev distance // Wikipedia.org. – Oct. 14, 2016. – https://en.wikipedia.org/wiki/Chebyshev_distance