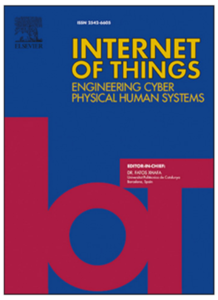


Cocv: A compression algorithm for time-series data with continuous constant values in IoT-based monitoring systems

Shengsheng Lin, Weiwei Lin, Keyi Wu, Songbo Wang, Minxian Xu, James Z. Wang

PII: S2542-6605(23)00372-4
DOI: https://doi.org/10.1016/j.iot.2023.101049
Reference: IOT 101049

To appear in: *Internet of Things*

Received date : 7 January 2023
Revised date : 19 September 2023
Accepted date : 22 December 2023

Cocv: A Compression Algorithm for Time-Series Data with Continuous Constant Values in IoT-based Monitoring Systems

Shengsheng Lin[a], Weiwei Lin[a,b,*], Keyi Wu[c], Songbo Wang[a], Minxian Xu[d], James Z. Wang[e]

[a]School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China
[b]Peng Cheng Laboratory, Shenzhen 518066, China
[c]South China Normal University, Guangzhou 510631, China
[d]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China
[e]School of Computing, Clemson University, SC, USA

Abstract

Sensor-generated time-series data now constitutes a significant and growing portion of the world's data due to the rapid proliferation of the Internet of Things (IoT). The transmission and storage of such voluminous data have emerged as enormous challenges. Data compression and reduction strategies have been instrumental in mitigating these challenges to some extent. However, they have exhibited limitations when applied to real-time IoT-based monitoring systems. This stems from their failure to adequately consider the stringent requirements of real-time data transmission and the continuous constant-value redundancy within periodic monitoring data. Consequently, we introduce a dedicated compression algorithm tailored specifically for time-series data within periodic IoT-based monitoring systems, namely Cocv. It takes advantage of the continuous constant-value repetition of the time-series data to compress data by discarding redundant data points. It can not only compress static batches of data but also dynamically compress data streams to improve system performance in real-time IoT-based monitoring systems. The offline Cocv outperforms traditional compressors on gas-leak monitoring data with a compression ratio of 98.5%, maintaining a decent speed for both compression and decompression. In an actual IoT-based gas-leak monitoring system, the online Cocv improves handling capacity by 255%, reading speed by 728%, reduces bandwidth consumption by 94%, and storage space consumption by 98% compared to the original scheme.

Keywords: Compression algorithm, Internet of things, Time-series data, Continuous constant values, gas-leak monitoring systems

1. Introduction

Thanks to the rapid proliferation of the Internet of Things (IoT) [1, 2], time-series applications have increasingly widespread, such as smart agriculture [3], anomaly detection [4], underwater wireless monitoring [5], and so on. In these IoT applications, monitoring systems are a class of systems with real-time transmission of monitoring data to obtain the latest status of monitored objects. Take the gas-leak monitoring system as an example [6, 7]. The gas-leak monitoring sensors laid in individual pipelines generate monitoring data points every few seconds, and these huge data generated day and night need to be transferred to the cloud server for processing and finally stored in time series database (TSDB). Huge network transmission bandwidth, server handling capacity, and disk storage space are required in this process. Therefore, how to reduce the cost of time-series data storage and transmission has become one of the concerns in the community.

Data compression serves as an effective way to mitigate the costs associated with the storage of time-series data, including both general-purpose and time-series-purpose compression techniques [8, 9]. However, data compression

*Corresponding author
Email addresses: `linss2000@foxmail.com` (Shengsheng Lin), `linww@scut.edu.cn` (Weiwei Lin), `18902306031@163.com` (Keyi Wu), `songbo1998@foxmail.com` (Songbo Wang), `mx.xu@siat.ac.cn` (Minxian Xu), `jzwang@clemson.edu` (James Z. Wang)

methods are primarily designed for the application at the end of storage (e.g., TSDB), as exemplified in Figure 1(a). More specifically, a batch of collected static data would be compressed via a designated algorithm and then stored in the TSDB [10, 11]. While this approach effectively minimizes disk space utilization, the transmission bandwidth consumption between IoT system modules remains restricted. To reduce the bandwidth consumption in the IoT system, data reduction solutions have been promoted in recent years [12]. Data reduction represents a technique aimed at preprocessing and reducing data prior to transmission, thereby reducing the frequency of communication within the IoT system. However, most existing data reduction techniques are implemented at the gateway or sensor side. These solutions are typically targeted at non-time-sensitive tasks, where the server allows the edge devices to accumulate a batch of data and then compress it before transferring it to the cloud server [13]. However, for systems like gas-leak monitoring systems, each data point needs to be transmitted in real-time to obtain timely status updates for the monitored object. Therefore, the real-time data transmission requirements of such systems constrain the application of existing data reduction techniques at the edge.

Additionally, it is essential to note that time series values within IoT-based monitoring systems tend to exhibit prolonged periods of slow change. For instance, in the context of gas leak monitoring, the values often persist at a constant 0 for extended durations, as gas leaks are relatively infrequent occurrences. Consequently, the data remain stationary for extended intervals, indicating the presence of a significant amount of redundant information. Regrettably, existing compression algorithms do not fully exploit this characteristic, thereby failing to optimize the compression ratio for such data. The literature [14] noticed this feature and utilized it to reduce the data transmission time between the server and gateway, but its usefulness is still limited in the gas-leak monitoring system due to the real-time data transmission requirements.

As a result, the existing compression and data reduction methods still have shortcomings for gas-leak monitoring systems. In this work, we propose a compression algorithm specifical for time-series data with continuous constant values (**Cocv**) in IoT-based monitoring systems to fill the gaps. It takes full account of the *continuous constant-value redundancy* that exists in monitoring data and achieves compression by *discarding superfluous redundant points* in the data stream, as shown in Figure 1(b). Cocv is available in offline and online versions, where the offline version can compress static batch data to achieve an extreme compression rate, and the online version can be applied to IoT-based monitoring systems to compress dynamic data streams in real-time. Specifically, compression with online Cocv is a dynamic and continuous process *on the server side*, rather than on the edge side in the previous studies. For instance, a data point will be recorded and discarded by the server when it is considered redundant to be compressed, so that it does not need to be transferred to TSDB and the users. This improvement can significantly reduce transmission frequency and bandwidth consumption, and thus greatly improve the performance of the entire application. Further, by pre-reducing the number of redundant points, other compression techniques or compressors can be overlaid on the storage side to achieve the ultimate compression ratio. Our experimental results show that offline Cocv outperforms the traditional compression algorithm, and the application of online Cocv greatly improves the gas-leak monitoring system's performance.

To summarize, the main contributions of this paper are as follows:

- We propose a specialized compression algorithm for time-series data with continuous constant values, achieved by discarding superfluous redundant data points.
- We mathematically prove that the proposed algorithm satisfies many desirable properties, including a high compression ratio, high computational efficiency, and lossless compression, when compressing periodic time-series data.
- We further refine the initially designed algorithm to an online version capable of operating on the server side. This adaptation has been successfully implemented in the IoT-based gas-leak monitoring system, leading to significant improvements in performance.
- Extensive experimental results based on both real-world and synthetic datasets show the proposed offline algorithm outperforms the traditional compression algorithm, and the application of its online version greatly improves the gas-leak monitoring system's performance.

The remainder of this paper is organized as follows: In Section 2, we conduct a comprehensive review of the related literature and present our findings. In Section 3, we introduce the offline Cocv algorithm and prove its desirable

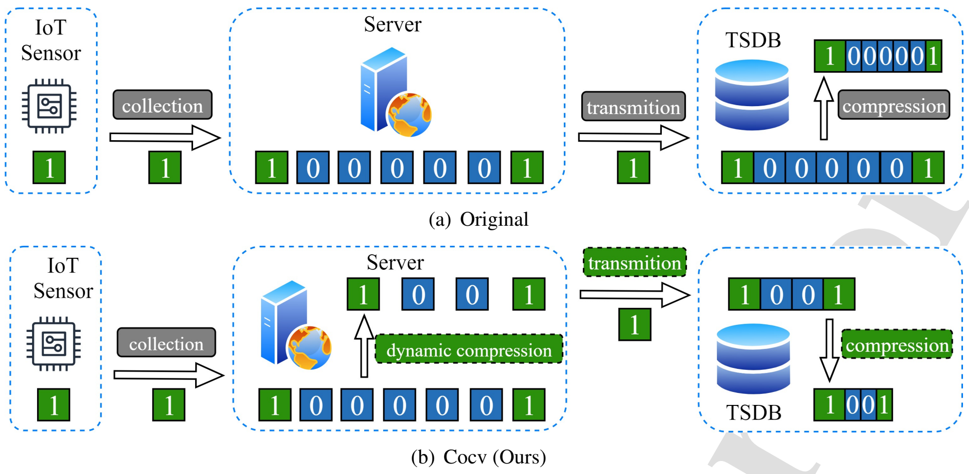


Figure 1: Comparison of the data flows in the IoT-based monitoring systems with different setups.

properties mathematically. Then in Section 4, we refine the offline Cocv to the online Cocv capable of operating on actual IoT-based monitoring systems. In Section 5, the detailed offline and online experiments of Cocv is presented. Finally, our work is summarized in Section 6.

2. Related work

Numerous research scholars have dedicated considerable effort to advancing the development of compression algorithms and data reduction techniques in recent decades [8, 12]. To distinguish our work effectively, we will provide a comprehensive overview of related research.

General and time-series compression algorithms. Traditional general-purpose compressors are characterized by compression algorithms based on dictionary coding [15, 16], which rely on the identification of shared identical segments within the data. Examples of such compressors include Gzip [17], Snappy [18], Lz4 [19], and Zstd [20]. Lossless compression of time-series data capitalizes on the distinctive features of time-series data for enhanced compression efficiency. Notable examples of dictionary-based compressors incorporating time-series properties are A-Lzss [21] and D-Lzw [22]. Additionally, Drh [23] and Sprintz [24] are sequential algorithms designed to compress time series data, employing a sequential combination of fundamental compression techniques, including Huffman coding [25], Delta coding [26], Run-length coding [27], among others. In recent years, machine learning techniques have also been explored for time series compression [28, 29, 30]. Dzip [31] stands out as a time series lossless compressor that employs deep neural networks [32] in conjunction with window prediction techniques. It endeavors to train a corresponding prediction model for the data slated for compression and subsequently retains the prediction error through arithmetic coding [33], thereby achieving lossless compression of time series data. While these data compression algorithms have made significant advancements in recent years, they have not been specifically tailored to address the characteristics of data, such as those found in gas leak monitoring, which exhibit a multitude of redundant data points. In contrast, Cocv capitalizes on precisely this attribute, accomplishing data compression by discarding superfluous redundant data points.

Data reduction techniques in IoT systems. The application of data reduction methods in IoT systems has proven its significance over the years [34, 35, 36]. It is worth noting that existing data reduction techniques typically operate at the edge of IoT systems, with the primary goal of conserving sensor energy and reducing data transmission overhead [37, 38, 39]. For instance, in wireless sensor networks (WSN) within the context of IoT [40], energy efficiency is a paramount concern due to the limited energy resources of sensor nodes. Consequently, numerous data reduction techniques for sensor nodes [41, 42, 43] have been developed, encompassing aggregation-based and compression-based methods. Additionally, several data reduction techniques are employed at the sensor level to reduce data transmission overhead [44, 45]. It is noteworthy that these methods often operate in scenarios where sensors can continuously collect data before central compression and reporting, which is not feasible in the context of gas leak monitoring. In gas leak scenarios, sensors must promptly report their latest status to the server to ensure safety, with no opportunity to accumulate data for later transmission. In this context, dynamic data stream reduction executed at the server side, as opposed to the traditional approach, holds significant promise. However, to the best of our knowledge, such methods are currently lacking, creating a gap in the domain of data reduction techniques within IoT systems.

In summary, Cocv distinguishes itself from existing compression algorithms and data reduction techniques by offering a compression algorithm for time-series data streams that excels in reducing redundant data points. The offline version of Cocv can be employed as a static compression algorithm for batch data compression. Simultaneously, the online version of Cocv facilitates dynamic data compression on the server side, effectively bridging the gap in data reduction techniques at the server level.

3. Offline compression algorithm for time-series data with continuous constant values

In this section, we first introduce the overview of Cocv for the offline version and then give the definitions of required notations and specific algorithm design. At the last, we prove its excellent properties, including high compression rate, low computational complexity, and lossless compression when compressing periodic time-series data.

3.1. Overview of offline Cocv

Using a formal notation [8], time series can be defined as:

$$TS = [(t_1, v_1), \dots, (t_n, v_n)], \qquad t_{i-1} < t_i < t_{i+1}, v_i \in \mathbb{R}, \tag{1}$$

where n is the number of data points and (t_i, v_i) is denoted as a time-series data point, of which t_i is the timestamp and v_i is the value. A time series can be sliced into multiple sequence segments. For such a sequence segment:

$$TS_{i,j} = [(t_i, v_i), \dots, (t_j, v_j)], \qquad j - i \geq 2, \tag{2}$$

if $\forall k \in [i, j)$, satisfy:

$$\begin{cases} v_k = v_i \\ t_k - t_{k-1} = t_{i+1} - t_i, \end{cases} \tag{3}$$

then we classify this segment as a *continuous constant-value segment* (CCS). In the case of an extended CCS, there exists significant redundancy within its data, allowing for a more concise representation. To be precise, it can be characterized by its *start time*, *fixed interval*, *end time*, and *constant value*, which can be used to reconstruct the same CCS. We observe that the complete description of the entire segment is inherently contained in the first, second, and last data points of the segment. Thus, theoretically, a CCS can be entirely represented using only these three data points.

As shown in Figure 2, the overview process of offline Cocv is described. During the compression process, superfluous redundant points are deduplicated, and only the segment's 1st, 2nd, and last data points are retained. In this way, we can achieve the compression of time-series data with CCSs. During the decompression process, the necessary information of the CCS (i.e., start time, fixed interval, end time, and constant value) is obtained from the 1st, 2nd, and last data points. And then the discarded data points can be recalculated back through them. Theoretically, the more the number and length of CCSs in a time-series data sequence, the better the compression ratio will be obtained.

3.2. Notations and algorithm design

Table 1 exhibits the notations in the offline Cocv. Algorithm 1 demonstrates the main process of offline Cocv, which accepts an uncompressed time-series data stream P and outputs a compressed time-series data stream R. Offline Cocv first initializes R to an empty array, and initializes f^v, f^t, f^c,f^i to *error*, 0, 0, 0 respectively. Line 1 of offline Cocv iterates through each data point c in the time-series data stream P. In line 2 of the algorithm, if $c^v = f^v$ and $c^t - f^t = f^i$, then it means that a CCS is found and the front point is in the middle of the segment, thus compression can be performed. Therefore, line 3 of the algorithm discards the original front point and performs f^c self-increment to record the length of the CCS. Line 4 implies that the current point is not in the CCS. In lines 5-7 of the algorithm,

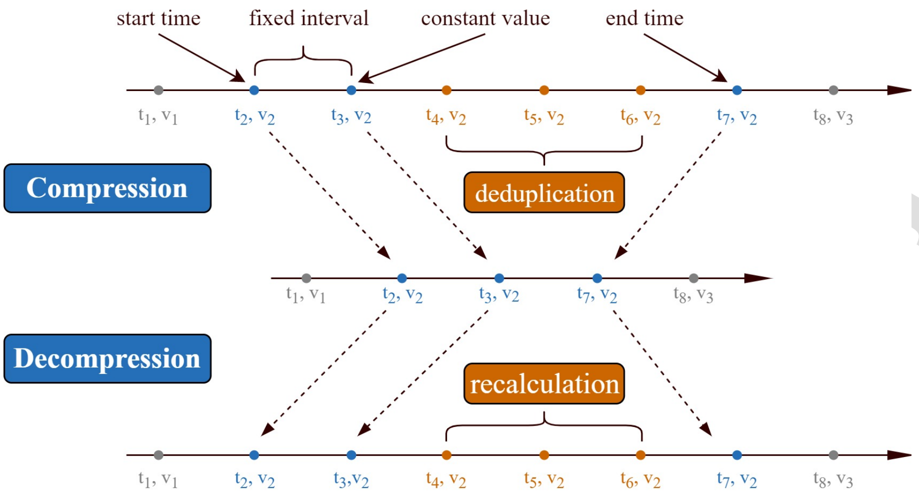


Figure 2: The overview process of offline Cocv compression and decompression.

Symbol	Description
P	Uncompressed time-series data stream
R	Compressed time-series data stream
D	Decompressed time-series data stream
c, c^t, c^v	Current data point to be compressed, and its timestamp and value
f, f^t, f^v, f^c, f^i	Front data point that was compressed, and its timestamp, value, times it has appeared consecutively, and the time interval
error	Signal of system error or outage (e.g., -2)
pointer	Pointer to the current data point when decompressing
interval	Time interval of the current data stream
R_i, R_i^t, R_i^v	The i-th data point in the compressed data stream, and its timestamp and value

Table 1: The notations of offline Cocv.

if $f^c > 0$, then it means that the original front point is the end of the previous CCS, so it needs to be pushed into the compressed data stream R. In lines 8-10, the current point is pushed into the compressed data stream R, and the value of f^v, f^t, f^c, f^i is updated to find the next CCS again. In lines 13-14, after the iteration of the time-series data stream P is completed, if $f^c > 0$, then it means that the last point of the data stream is just the end of the last CCS, so that point is pushed into the compressed data stream. Algorithm 1 finally returns the compressed time-series data stream R and ends.

Algorithm 2 shows the main process of decompression of offline Cocv, which accepts the compressed time-series data stream R and outputs the decompressed time-series data stream D. It first initializes pointer to 0. In lines 1 and 2 of the algorithm, if the length of the time-series data stream R is less than or equal to 2, then it means that there must be no CCS in this data stream (according to Eqn.2, the minimum length of a CCS is 3) and can directly return without decompression. In lines 4 and 5 of the algorithm, the difference between the timestamps of the 1th and 0th data points is recorded as *interval*, and these two points are directly pushed into the decompressed result stream D. In line 6 of the algorithm, the first to the penultimate data point in the compressed time-series data stream R is traversed. In lines 7 and 8 of the algorithm, if the next data point is not equal to the value of the current data point, or its timestamp difference modulo *interval* is not equal to 0, then it means that the next data point is not an endpoint of a CCS, so it is directly pushed into the decompression result data stream D. Lines 9-13 imply that the next point and the current point are the endpoints of a CCS, so the original discarded data points can be recalculated through *interval* and the information of these two points. Line 16 of the algorithm recalculates the new *interval*. Algorithm 2 finally returns

Algorithm 1 Offline compression algorithm for time-series data with continuous constant values

Input: Uncompressed time-series data stream P.
Output: Compressed time-series data stream R.
Init: $R, f^v, f^t, f^c, f^i \leftarrow \emptyset, error, 0, 0, 0.$

```
for $c$ in $P$ do        /* Iterates through the Uncompressed stream */
  if $c^v = f^v$ and $c^t - f^t = f^i$ then        /* A CCS is found and perform compression */
    $f^t, f^c \leftarrow c^t, f^c + 1$
  else
    if $f^c > 0$ then        /* The end of the last CCS */
      $R \leftarrow R \cup f$
    end if
    $R \leftarrow R \cup c$        /* The begin of the following CCS */
    $f^i \leftarrow c^t - f^t$        /* Updates variables */
    $f^v, f^t, f^c \leftarrow c^v, c^t, 0$
  end if
end for
if $f^c > 0$ then        /* The end of the endmost CCS */
  $R \leftarrow R \cup f$
end if
```

Algorithm 2 Offline decompression algorithm for time-series data with continuous constant values

Input: Compressed time-series data stream R.
Output: Decompressed time-series data stream D.
Init: $pointer \leftarrow 0.$

```
if $|R| \leq 2$ then        /* No compression required */
  $D \leftarrow R$
else
  $interval \leftarrow R^t_1 - R^t_0$        /* Gets interval */
  $D \leftarrow D \cup R_0 \cup R_i$
  for $i$ in $1, \dots, |R| - 2$ do        /* Executes decompression */
    if $R^v_{i+1} \neq R^v_i$ or $\left(R^t_{i+1} - R^t_i\right) \,\%\, interval \neq 0$ then        /* Finds a discrete point */
      $D \leftarrow D \cup R_{i+1}$
    else
      $pointer \leftarrow R^t_i$
      while $pointer \neq R^t_{i+1}$ do        /* Finds a compressed CCS and recover it. */
        $pointer \leftarrow pointer + interval$
        $D \leftarrow D \cup \left(pointer, R^v_i\right)$
      end while
    end if
    $interval \leftarrow R^t_{i+1} - R^t_i$        /* Updates interval */
  end for
end if
```

the decompressed time-series data stream D and ends.

3.3. Desirable properties

Offline Cocv exhibits desirable properties, including high compression rate, low computational complexity, and lossless compression, when compressing periodic time-series data. We will prove it in the following.

Theorem 1. *Offline Cocv exhibits a high compression ratio applicable to periodic time-series data featuring a substantial number of CCSs.*

Proof of Theorem 1. To better describe the compression ratio, we define the compression score CS here:

$$CS = Compression\_Score = (1 - \frac{Compressed\_Size}{Origin\_Size}) \times 100. \tag{4}$$

For a time series, we define L as the total length of the series, N as the number of CCSs inside the time series that satisfy Eqn.3, Avg_Len as the average length of these CCSs, and Dis_Rate as the total proportion of discontinuous points that not in those CCSs. For the CCSs with a regular interval, the first segment will be compressed to 3 data points, while the other segments that follow will be compressed to 2 points. Therefore, we have:

$$Compressed\_Size = L \times Dis\_Rate + 2 \times N + 1. \tag{5}$$

And then the compression score for this time series can be expressed as:

$$CS = (1 - \frac{L \times Dis\_Rate + 2 \times N + 1}{L}) \times 100. \tag{6}$$

In addition, the identity of the total points number for the CCSs can be described as:

$$N \times Avg\_Len = L \times (1 - Dis\_Rate). \tag{7}$$

Substituting it into Eqn.6, we have:

$$CS = (1 - \frac{L \times Dis\_Rate + 2 \times \frac{L\times(1-Dis\_Rate)}{Avg\_Len} + 1}{L}) \times 100. \tag{8}$$

After simplifying it, we have:

$$CS = ((1 - Dis\_Rate) \times \frac{Avg\_Len - 2}{Avg\_Len} + \frac{1}{L}) \times 100. \tag{9}$$

When the length L of this time series tends to infinity, i.e., $1/L$ tends to infinity small, we have:

$$CS \propto Avg\_Len, \tag{10}$$

and

$$CS \propto \frac{1}{Dis\_Rate}. \tag{11}$$

When compressing periodic time-series data characterized by a significant count of CCSs, specifically exhibiting a low value of Dis_Rate and a substantial Avg_Len, the Cocv compression method demonstrates remarkable efficiency in achieving substantial compression ratios.

Theorem 2. *Offline Cocv exhibits low computational complexity.*

Proof of Theorem 2. In Algorithm 1, lines 2-11 handle the data points within the time-series data stream with an upper complexity bound of $O(1)$. However, line 1 of the algorithm traverses the time-series data stream denoted as P, resulting in a complexity of $O(|P|)$ for lines 1-12 collectively. Lines 13-15 involve only simple operations and have a constant complexity of $O(1)$. Therefore, the overall complexity of Algorithm 1 is determined by $O(|P|)$, which can be equivalently expressed as $O(L)$.

Turning to Algorithm 2, lines 11-14 represent the most computationally intensive portion. The complexity of lines 11-14 is tied to the length of the CCS and is expressed as $O(|\frac{R^t_{i+1}-R^t_i}{Interval}|)$. This complexity scales relative to the length of the CCS. If we denote the average CCS length as Avg_Len, then $O(|\frac{R^t_{i+1}-R^t_i}{Interval}|) = O(Avg\_Len)$. Additionally, line 6 of the algorithm has a complexity of $O(|R|)$, equivalently $O(Compressed\_Size)$, while the remaining lines exhibit constant complexity, i.e., $O(1)$. Consequently, the overall complexity of Algorithm 2 can be expressed as $O(Avg\_Len \times Compressed\_Size)$. By combining Eqn.4 and Eqn.10, we establish that $Avg\_Len \times Compressed\_Size$ is proportional to L, implying that the complexity of Algorithm 2 scales as $O(L)$.

The compression and decompression algorithms of offline Cocv exhibit a computational complexity of $O(L)$ each. This linear relationship with the length of the time-series data sequence indicates that both compression and decompression operations have low computational demands.

Theorem 3. *Offline Cocv constitutes a lossless compressor applicable to periodic time-series data.*

Proof of Theorem 3. A time-series sequence can be partitioned into two distinct components: CCSs with a length greater than 2 and discrete data points not belonging to these segments. In the context of regular interval time-series sequences, it is theoretically possible to compress all CCSs into just two discrete endpoints, with the exception of the first segment, which requires compression to three points. Simultaneously, all discrete points outside of a CCS remain unaltered. Consequently, the original data transforms into a sequence of discrete points devoid of any initial CCSs.

However, in the extreme case of time-series sequences featuring irregular intervals, it is conceivable that discrete points might coincide exactly with the endpoints of nonexistent CCSs. In such instances, during the decompression process, these particular discrete data points will be erroneously interpreted as endpoints of CCSs, inadvertently reintroducing non-existent CCSs. In this scenario, offline Cocv cannot be considered strictly lossless compression, as it cannot ensure the consistency of the decompressed data with the original data. Nevertheless, this can be construed as a linear interpolation to compensate for missing information in practical IoT-based systems, thereby exerting no significant impact on the system's operational information.

When dealing with time-series data characterized by fixed equal intervals, Cocv can indeed guarantee lossless compression and decompression. Each segment generated by the offline Cocv decompression algorithm unquestionably originates from a CCS in the original data. This guarantees the lossless compression attribute of offline Cocv in cases where the time-series sequence to be compressed adheres to a fixed, equal time interval.

In summary, offline Cocv has the properties of high compression rate, low computational complexity, and lossless compression applicable to periodic time-series data.

4. Online compression algorithm for time-series data with continuous constant values

Offline Cocv demonstrates remarkable compression capabilities for offline data, but the objective of this research extends beyond solely mitigating disk storage overhead for continuous constant-value-type time-series data. It also aims to enhance the overall performance of IoT-based monitoring systems, encompassing aspects such as handling capacity, data reading time, and bandwidth consumption. Consequently, we refine offline Cocv into an online iteration, tailored to fulfill the demands of real-time compression within IoT-based monitoring systems. It is noteworthy that the implementation of online Cocv will reside on the system server side.

We take the gas-leak monitoring system as an example to first introduce the overall architecture and features of the IoT-based monitoring systems and then describe the improved online Cocv compression algorithm.

4.1. Architecture of IoT-based monitoring system

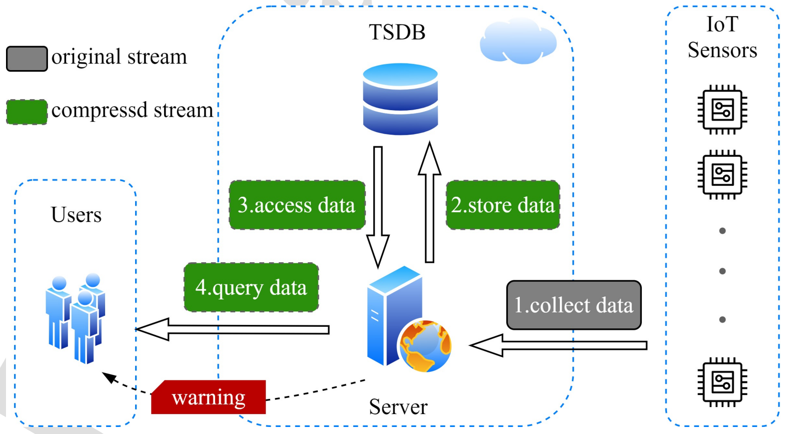


Figure 3: A typical architecture of an IoT-based gas-leak monitoring system and its optimized data streams compressed by Cocv.

As shown in Figure 3, a typical gas-leak monitoring system comprises a multitude of sensors, a centralized or distributed server, and a TSDB. Each sensor collects gas-leak monitoring information from the gas pipe at regular intervals (e.g., every 10 seconds) and uploads it to the server. The server accepts and analyzes the data uploaded by these sensors. If the data exhibits abnormality, an early warning notification would be sent to the users. TSDB accepts the processed data from the server and stores it on the disk. Users query the monitoring data through the server to query the data stored in TSDB. Usually, the data streams transmitted between different components would consume huge communication bandwidth and occupy a huge handling capacity of the server before the system architecture was optimized.

After being optimized by Cocv, the compressed data streams in the IoT-based monitoring system are also shown in Figure 3, including the data streams transmitted between the server, TSDB, and users. Specifically, a data point will be recorded and discarded by the server when it is considered redundant to be stored, so that it does not need to be transferred to TSDB. Similarly, only a small amount of reduced data needs to be transferred when the users query and the server access the data. The data would be restored by the low-complexity decompression algorithm of Cocv, which might be a process that will be much faster than reading large amounts of raw data directly. In that way, the server's handling capacity, users' reading time, TSDB's disk consumption, and bandwidth consumption will gain significant improvements.

Nonetheless, there exist three challenges within the gas-leak monitoring system that preclude the direct application of offline Cocv:

(1) **Parallel Data Arrival**: Due to parallelization, data from multiple sensors may arrive simultaneously. Consequently, the algorithm necessitates refinement to enable concurrent compression of multiple time series.
(2) **Sensor Stability**: The stability of individual sensors cannot be assured. Each sensor may abruptly go offline due to hardware or network issues. To address this concern, Cocv requires the integration of a sensor offline monitoring module, enhancing the overall system's data integrity. It is worth noting that this module is optional; the algorithm can still function correctly without it, albeit at the cost of missing sensor offline information.
(3) **Server Stability**: Similarly, the stability of the server cannot be guaranteed. The server may experience downtime at any given moment, leading to a disruption in the Cocv algorithm, resulting in significant information loss. Consequently, online Cocv must incorporate mechanisms to recover lost information following server interruption and restart.

To accommodate these three challenges, the optimized online compression algorithm will be introduced in the next section. Notably, the corresponding decompression algorithm is directly derived from Algorithm 2, but it is crucial to emphasize that the compressed data from each sensor necessitates separate decompression.

4.2. Notations and algorithm design

Table 2 exhibits the notations in the online Cocv. Algorithm 3 shows the main process of online Cocv, which accepts the sensor set S and the uncompressed time-series data stream P, and then outputs the compressed time-series data stream R. In line 1 of the algorithm, each sensor in the sensor set S is traversed. In line 2 of the algorithm, if $f_s = null$, then it means that the sensor s is not initialized, so line 3 initializes it. Line 4 of the algorithm indicates that sensor s has been initialized, which further indicates that the server has been offline previously. Therefore, lines 5 and 6 of the algorithm push the former information into the time-series data stream R and reinitialize the information of the sensors. It should be noted that the information of f_s^v, f_s^t, f_s^c, f_s^i can be stored after the server going offline with the help of persistence techniques of Redis [46] or other persistence techniques. The sensor offline detection in line 9 is an optional module and is described in the description of Algorithm 4. Lines 10-26 of Algorithm 3 essentially exhibit a comparable compression process to that of lines 1-15 in Algorithm 1. Specifically, this involves the identification and removal of redundant data points within CCSs. They differ in that Algorithm 3 distinguishes which sensor the current data point belongs to, thus enabling parallelized compression of multiple time series.

Algorithm 4 shows the main process of the sensor offline detection algorithm, which accepts the sensor set S and the fixed interval $fixed$ to detect the sensor offline, and outputs the offline monitoring data stream P to Algorithm 3. In line 1 of the algorithm, the thread sleeps for $fixed$ to wait for the detection. In lines 2-4 of the algorithm, a new thread is executed to monitor the sensor's offline status after each sleep for $fixed$. In line 5 of the algorithm, each sensor in the sensor set S is traversed. In lines 6 and 7 of the algorithm, if the front data timestamp exceeds now $2 \times fixed$ or

Symbol	Description
S, s	Set of sensors and its sensor
P	Uncompressed time-series data stream
R	Compressed time-series data stream
D	Decompressed time-series data stream
c_s, c_s^t, c_s^v	Current data point of the sensor s to be compressed, and its timestamp and value
$f_s, f_s^t, f_s^v, f_s^c, f_s^i$	Front data point of sensor s that was compressed, and its timestamp, value, times it has appeared consecutively, and the time interval from the previous point
$error$	Signal of system error or outage (e.g.,-2)
$offline$	Signal of sensor offline (e.g.,-1)
now	Current system time
$fixed$	Fixed interval to detect sensor offline
$pointer_s$	Pointer to the current data point of sensor s when decompressing
$interval_s$	Time interval of the current data stream of sensor s
$R_{s_i}, R_{s_i}^t, R_{s_i}^v$	The i-th data point of sensor s that was compressed, and its timestamp and value

Table 2: The notations of online Cocv.

Algorithm 3 Online compression algorithm for time-series data with continuous constant values

```
Input: Set of sensors $S$, uncompressed time-series data stream $P$.
Output: Compressed time-series data stream $R$.
1: for $s$ in $S$ do
2:   if $f_s = null$ then          /* Initialize uninitialized sensors */
3:     $f_s^v, f_s^t, f_s^c, f_s^i \leftarrow error, now, 0, 0$
4:   else          /* Recording error and reinitialize sensors */
5:     $R \leftarrow R \cup f_s \cup (f_s^t + 1, error)$
6:     $f_s^v, f_s^t, f_s^c, f_s^i \leftarrow error, now - 1, 0, 0$
7:   end if
8: end for
9: execute sensor offline detection (optional)
10: for $c_s$ in $P$ do          /* Executes Cocv's compression process */
11:   if $c_s^v = f_s^v$ and $c_s^t - f_s^t = f_s^i$ then
12:     $f_S^t, f_s^c \leftarrow c_s^t, f_s^c + 1$
13:   else
14:     if $f_s^c > 0$ then
15:       $R_s \leftarrow R_s \cup f_s$
16:     end if
17:     $R_s \leftarrow R_s \cup c_s$
18:     $f_s^i \leftarrow c_s^t - f_s^t$
19:     $f_s^v, f_s^t, f_s^c \leftarrow c_s^v, c_s^t, 0$
20:   end if
21: end for
22: for $s$ in $S$ do          /* The end of compression process */
23:   if $f_s^c > 0$ then
24:     $R \leftarrow R \cup f$
25:   end if
26: end for
```

the last data point is offline, then it means that the sensor s is offline now, so it sends the offline information of this sensor to the stream P.

Algorithm 4 Detection algorithm for offline sensor (optional)

Input: Set of sensors S, the fixed interval $fixed$ to detect sensor offline.
Output: The offline monitoring data stream P.

```
sleep($fixed$)
while true do        /* Performs offline detection every $fixed$ */
  sleep($fixed$)
  new Thread do
  for $s$ in $S$ do
    if $f_s^t - now \geq 2 \times fixed$ or $f_s^v = offline$ then        /* Detects offline and sends signal to main program */
      $P_s \leftarrow P_s \cup (now, offline)$
    end if
  end for
end while
```

5. Experimental results and discussion

The performance of Cocv is evaluated by experimenting with real data sets and simulated generated data in this section.

5.1. Experimental setup

5.1.1. Comparison algorithm

The following comparison algorithms are employed to assess the performance of Cocv. These algorithms consist of four general-purpose compressors, one sequential time-series compressor, and one neural network-based compression technique.

Gzip [17]: A lossless compression algorithm built on the Deflate methodology. It typically offers a high compression ratio at the expense of slower compression speeds.

Snappy [18]: A lossless compression algorithm designed for rapid compression with an acceptable compression ratio.

Lz4 [19]: A lossless compression algorithm optimized for swift compression and decompression operations.

Zstd [20]: A compression algorithm employing Finite State Entropy (FSE), enabling the adjustment of compression levels from -7 (fastest speed) to 22 (highest compression ratio).

Delta [26]: Utilizes double delta coding on timestamps to compress time-series data.

Dzip [31]: A lossless compression algorithm that incorporates deep neural networks and window prediction techniques.

Cocv_Zstd: A variant of Cocv, which involves applying the Zstd compression algorithm after Cocv. This approach has the potential to achieve an even more remarkable compression ratio.

5.1.2. Evaluation metrics

The following defined metrics are employed to evaluate the performance of the proposed algorithm in this paper:
Compression Score: This metric quantifies the compression ratio.

$$Compression\_Score = (1 - \frac{Compressed\_Size}{Origin\_Size}) \times 100. \tag{12}$$

Compression Speed: This metric measures the compression speed.

$$Compression\_Speed = \frac{Origin\_Size}{Compression\_Time}. \tag{13}$$

Decompression Speed: This metric evaluates the decompression speed.

$$Decompression\_Speed = \frac{Origin\_Size}{Decompression\_Time}. \tag{14}$$

Handling Capacity: This metric gauges the server's capability to process sensor data uploads per unit of time.

$$Handling\_Capacity = \frac{Origin\_Points\_Number}{Handling\_Time}. \tag{15}$$

Reading Speed: This metric measures user retrieval speed.

$$Reading\_Speed = \frac{Origin\_Points\_Number}{Reading\_Time}. \tag{16}$$

Bandwidth Consumption: This metric assesses the bandwidth consumption between the server and TSDB.

$$Bandwidth\_Consumption = \frac{Transmission\_Points\_Number}{Handling\_Time}. \tag{17}$$

Storage Consumption: This metric quantifies the TSDB storage space usage.

$$Storage\_Consumption = Storage\_Points\_Number. \tag{18}$$

5.1.3. Dataset

Real Data: We obtained real data from a gas-leak monitoring company in China, including measurements from 100 gas-leak monitoring sensors over a period of one month. The information of this dataset is recorded in Table 3.

Size of origin data (KB)	Total number of points	Number of CCSs	Average length of CCSs
121,082	8,242,285	40,700	201

Table 3: The information of the real gas-leak monitoring data.

Synthetic Data: To better evaluate the performance of Cocv, simulated data is also employed. We generated 399 data files in which the rate of discontinuous points rises from 0 to 0.5 and the average length of CCSs rises from 4 to 1000, with each data file containing 864,000 data points.

5.1.4. Experimental environment

The experiments in this section were performed on a CentOS server configured with Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz, 256GB RAM, along with TSDB service (10,000 transactions per second) and Redis components [46]. The experimentation process involved the simulation of a gas-leak monitoring system as illustrated in Figure 3. This simulation entailed the deployment and utilization of 100 gas-leak monitoring sensors.

5.2. Experimental result of offline Cocv

Figure 4 illustrates the compression performance of the offline Cocv algorithm and several comparative algorithms when applied to real gas-leak monitoring data. The findings demonstrate that offline Cocv attains a top-tier compression ratio, surpassing conventional general-purpose compression algorithms such as Gzip, Lz4, and Snappy by a significant margin, all while maintaining a reasonable compression speed. Zstd, benefiting from the novel finite-state entropy technique, also achieves commendable results in terms of compression ratio and compression speed. In the case of Delta, which serves as a sequential compression algorithm designed for time-series data, it fails to achieve a favorable compression ratio for this particular data type. This may be attributed to Delta's necessity to store each data point even after size reduction, resulting in minimal space savings in comparison to the substantial redundancy present in the data. Concerning Dzip, a deep learning data compression framework, it likewise attains an almost ideal compression ratio. Nonetheless, questions persist regarding its practical applicability due to its unsatisfactory compression and decompression speed. This drawback arises from the considerable time required for training a matching model tailored to the data intended for compression.

Overall, Cocv achieves a near-optimal compression ratio while maintaining a decent compression and decompression speed that meets practical needs compared with these algorithms. In addition, by stacking Cocv and Zstd, a more

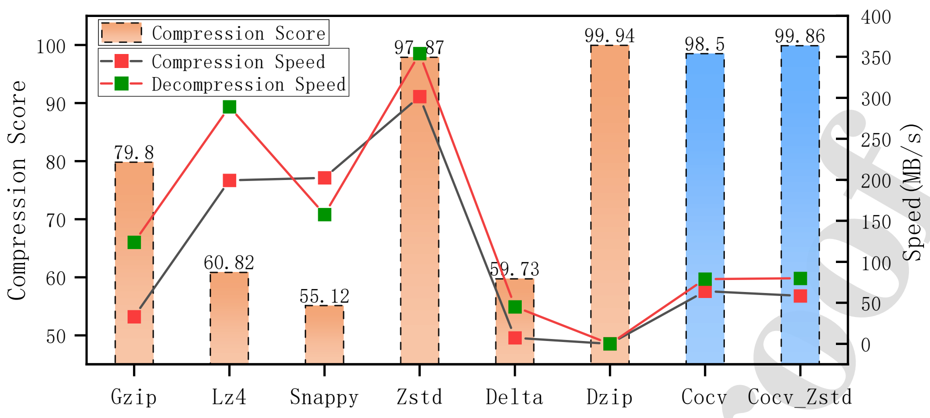
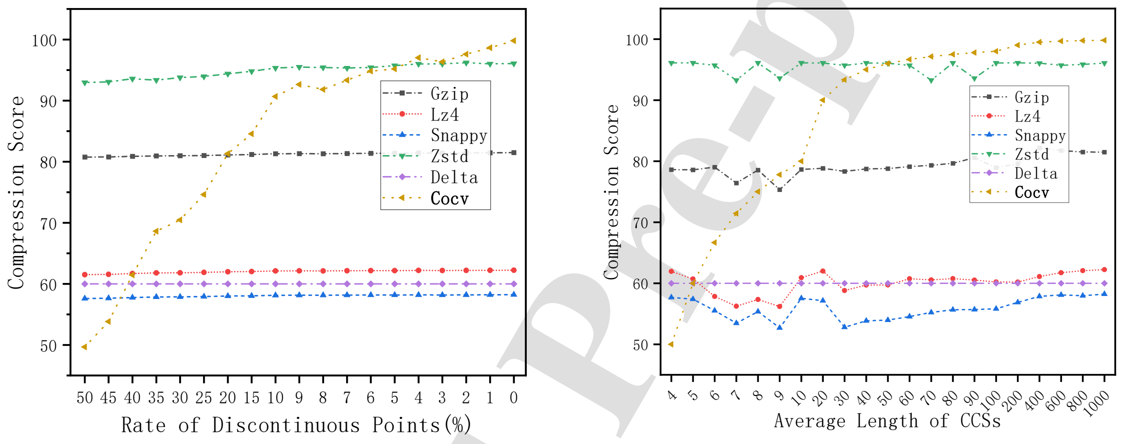


Figure 4: Compression performance of offline Cocv and other algorithms on the real data collected from a Chinese gas-leak monitoring company.



(a) Impact of the Rate of Discontinuous Points

(b) Impact of Average Length of CCSs

Figure 5: Compression score of offline Cocv and other comparison algorithms on the synthetic data.

extreme compression ratio can be achieved without much speed loss. Not only Zstd but also any other algorithms can be overlaid with the Cocv algorithm to achieve higher compression ratios according to the actual requirement.

The results of Figure 5 show the effect of the compression ratio of offline Cocv and other compared algorithms as the rate of discontinuous points and the average length of CCSs change. Figure 6 shows the effect of the compression ratio of offline Cocv as the rate of discontinuous points and the average length of CCSs change simultaneously. It can be found that as the rate of discontinuous points decreases and the average length of CCSs increases, the compression score of offline Cocv increases, while the other compared algorithms remain the same or only increase a little. Specifically, Cocv will outperform most of the comparison algorithms when the rate of discontinuous points is below 10% and the average length of continuous segments exceeds 100. Therefore, Cocv can perform satisfactorily in the face of time-series data with obvious constant-value continuity, such as gas-leak monitoring data, temperature sensing data, disk usage, and other time-series data with abundant continuous-constant values.

5.3. Experimental result of online Cocv

We have applied the enhanced online Cocv methodology to an actual IoT-based gas-leak monitoring system, which comprises a TSDB with a maximum write capacity of 10,000 transactions per second (TPS) and a Redis component. The performance improvements resulting from the implementation of online Cocv are presented in Table 4. Notably,

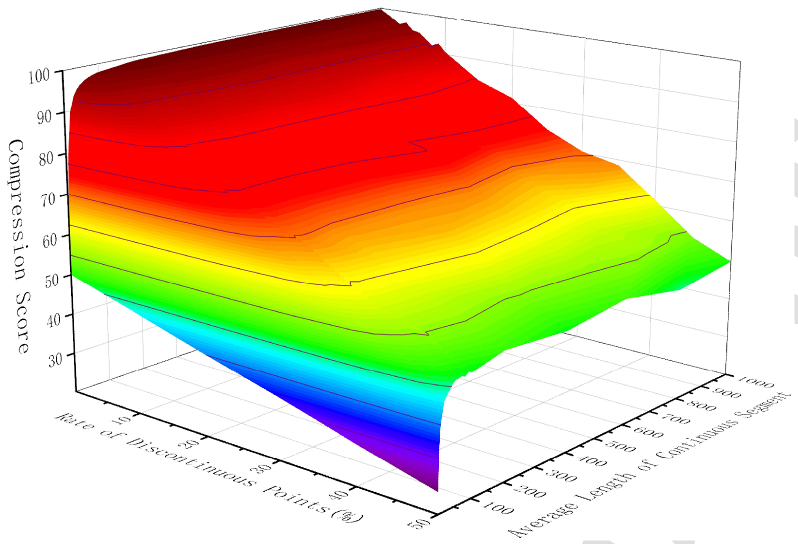


Figure 6: Compression score of offline Cocv with different rates of discontinuous points (rise from 0% to 50%) and average length of continuous segments (rise from 4 to 1000).

	Handling Capacity	Reading Speed	Bandwidth Consumption	Storage Consumption
Original Scheme	9,036	269,982	9,036	8,242,285
Online Cocv	32,102	2,235,499	479	12,3621
Improve	255%	728%	94%	98%

Table 4: The improvement achieved by the online Cocv over the original scheme.

the online Cocv solution has demonstrated substantial enhancements in various aspects of the system's performance, encompassing handling capacity, reading speed, bandwidth consumption, and storage consumption.

Furthermore, Figure 7 provides a detailed illustration of the performance outcomes under different system configurations. In a synchronous setup, online Cocv exhibits a remarkable 108× increase in handling capacity and a 6× enhancement in reading speed compared to the original scheme. Importantly, it maintains nearly constant bandwidth consumption while achieving a substantial 98% reduction in storage consumption. This observed performance gain is primarily attributed to the synchronous mode's predominant network bandwidth bottleneck, where Cocv's impact on bandwidth consumption remains relatively consistent with the original scheme. Nevertheless, substantial improvements are observed in other performance metrics.

Conversely, when operating in asynchronous mode, the system bottleneck shifts to the single-threaded CPU. Under these conditions, online Cocv enhances the handling capacity by 58%, reading speed by 722%, reduces bandwidth consumption by 97%, and lowers storage consumption by 98% compared to the original scheme. In pursuit of optimizing system performance, IoT-based systems generally opt for asynchronous multi-threaded operation. In this mode, the original approach experiences a bottleneck in the write speed of the TSDB, while the online Cocv approach encounters a bottleneck in CPU execution speed. Consequently, online Cocv yields substantial improvements, including a 255% increase in handling capacity, a 728% improvement in reading speed, a 94% reduction in bandwidth consumption, and a 98% decrease in storage consumption when compared to the original scheme. As a result, online Cocv emerges as a powerful tool for significantly enhancing the performance of IoT-based monitoring systems.

Consequently, it is evident that both offline Cocv and online Cocv exhibit satisfactory performance. Offline Cocv yields exceptional compression ratios and high compression speeds when applied to offline batch data. Moreover, the utilization of online Cocv in IoT-based monitoring systems leads to a significant enhancement in performance.

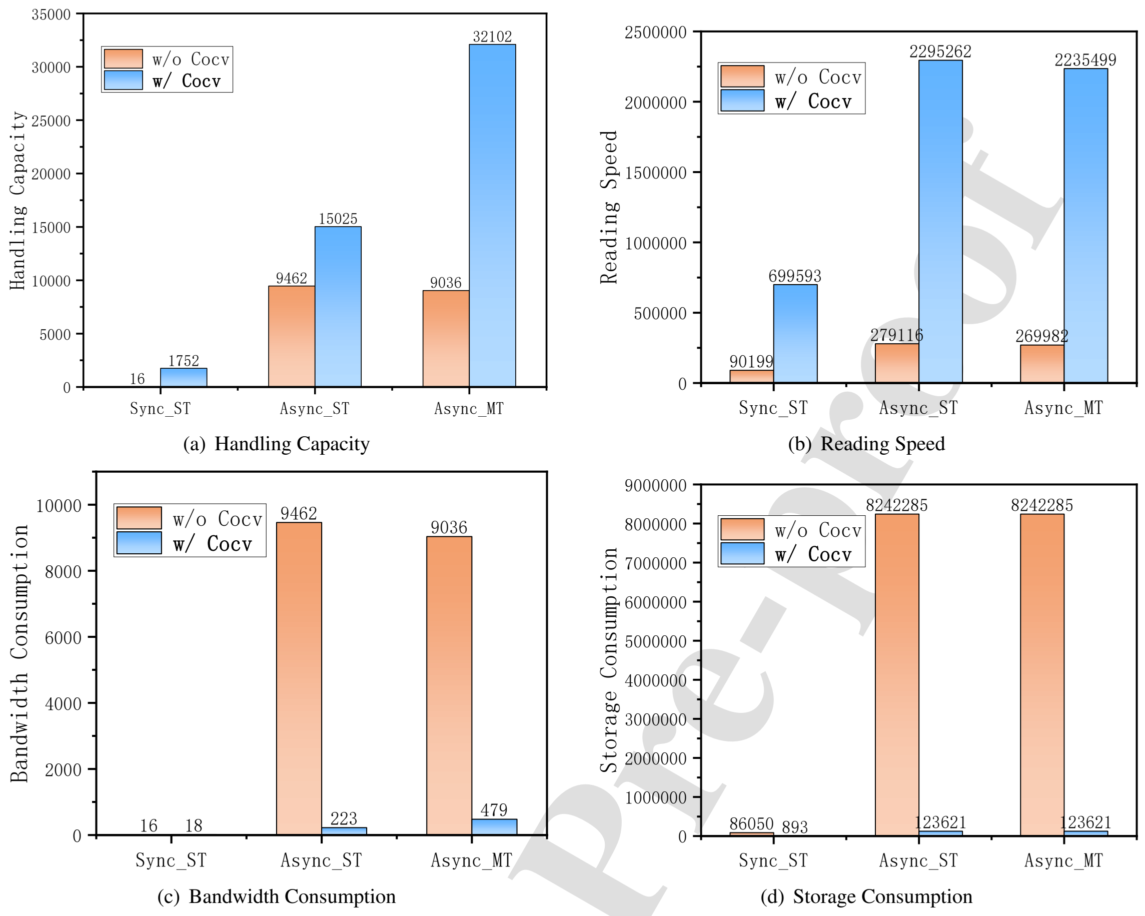


Figure 7: The evaluation of IoT-based monitoring systems' performance with various system configurations: (i) Synchronous Single-Threading Mode (Sync_ST), (ii) Asynchronous Single-Threading Mode (Async_ST), and (iii) Asynchronous Multithreading Mode (Async_MT).

6. Conclusion and future work

In this paper, we propose Cocv, a compression algorithm for time-series data with continuous constant values in IoT-based monitoring systems. Cocv is designed to improve the performance of IoT-based monitoring systems by reducing redundancy in the time-series data. Cocv satisfies many desirable properties, including a high compression ratio, high computational efficiency, and lossless compression for time-series data with a regular time interval. In the offline scenario, Cocv achieves a compression ratio of 98.5%, which substantially outperforms the traditional general-purpose compressors. In the online scenario of a periodic IoT-based gas-leak monitoring system, Cocv improves handling capacity by 255%, reading speed by 728%, reduces bandwidth consumption by 94%, and storage space consumption by 98% compared to the original scheme. In future work, we aspire to extend Cocv to non-periodic IoT-based monitoring systems, aiming to enhance the practical effectiveness of Cocv. Furthermore, exploring the application of Cocv on the sensor side is also a promising direction worth investigating.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work is supported by National Natural Science Foundation of China (620721878), Guangdong Special Project for Marine Economy Development (GDNRC[2022]17), Guangzhou Development Zone Science and Technology Project (2021GH10), and Major Key Project of PCL (PCL2021A09).

References

[1] H. K. Apat, R. Nayak, B. Sahoo, A comprehensive review on internet of things application placement in fog computing environment, Internet of Things 23 (2023) 100866. doi:https://doi.org/10.1016/j.iot.2023.100866.
URL `https://www.sciencedirect.com/science/article/pii/S2542660523001890`

[2] A. A. Laghari, K. Wu, R. A. Laghari, M. Ali, A. A. Khan, A review and state of art of internet of things (iot), Archives of Computational Methods in Engineering (2021) 1–19.

[3] A. K. M. Al-Qurabat, Z. A. Mohammed, Z. J. Hussein, Data traffic management based on compression and mdl techniques for smart agriculture in iot, Wireless Personal Communications 120 (3) (2021) 2227–2258.

[4] W. Wu, L. He, W. Lin, Y. Su, Y. Cui, C. Maple, S. Jarvis, Developing an unsupervised real-time anomaly detection scheme for time series with multi-seasonality, IEEE Transactions on Knowledge and Data Engineering 34 (9) (2022) 4147–4160. doi:10.1109/TKDE.2020.3035685.

[5] G. A. M. Jawad, A. K. M. Al-Qurabat, A. K. Idrees, Maximizing the underwater wireless sensor networks' lifespan using btc and mnp5 compression techniques, Annals of Telecommunications (2022) 1–21.

[6] M. Meribout, Gas leak-detection and measurement systems: Prospects and future trends, IEEE Transactions on Instrumentation and Measurement 70 (2021) 1–13. doi:10.1109/TIM.2021.3096596.

[7] L. Dong, Z. Qiao, H. Wang, W. Yang, W. Zhao, K. Xu, G. Wang, L. Zhao, H. Yan, The gas leak detection based on a wireless monitoring system, IEEE Transactions on Industrial Informatics 15 (12) (2019) 6240–6251. doi:10.1109/TII.2019.2891521.

[8] G. Chiarot, C. Silvestri, Time series compression survey, ACM Comput. Surv. 55 (10) (feb 2023). doi:10.1145/3560814.
URL `https://doi.org/10.1145/3560814`

[9] U. Jayasankar, V. Thirumal, D. Ponnurangam, A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications, Journal of King Saud University - Computer and Information Sciences 33 (2) (2021) 119–140. doi:https://doi.org/10.1016/j.jksuci.2018.05.006.

[10] C. Wang, J. Qiao, X. Huang, S. Song, H. Hou, T. Jiang, L. Rui, J. Wang, J. Sun, Apache iotdb: A time series database for iot applications, Proc. ACM Manag. Data 1 (2) (jun 2023). doi:10.1145/3589775.
URL `https://doi.org/10.1145/3589775`

[11] J. Xiao, Y. Huang, C. Hu, S. Song, X. Huang, J. Wang, Time series data encoding for efficient storage: A comparative analysis in apache iotdb, Proc. VLDB Endow. 15 (10) (2022) 2148–2160. doi:10.14778/3547305.3547319.
URL `https://doi.org/10.14778/3547305.3547319`

[12] L. Pioli, C. F. Dorneles, D. D. de Macedo, M. A. Dantas, An overview of data reduction solutions at the edge of iot systems: a systematic mapping of the literature, Computing (2022) 1–23.

[13] I. D. I. Saeedi, A. K. M. Al-Qurabat, Perceptually important points-based data aggregation method for wireless sensor networks, Baghdad Science Journal 19 (4) (2022) 0875–0875.

[14] K. Sari, M. Riasetiawan, The implementation of timestamp, bitmap and rake algorithm on data compression and data transmission from iot to cloud, in: 2018 4th International Conference on Science and Technology (ICST), 2018, pp. 1–6. doi:10.1109/ICSTC.2018.8528698.

[15] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Transactions on information theory 23 (3) (1977) 337–343.

[16] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE transactions on Information Theory 24 (5) (1978) 530–536.

[17] G. Jean-loup, A. Mark, The gzip home page, `https://www.gzip.org/`, accessed September 16, 2023 (2003).

[18] H. G. Steinar, Snappy — a fast compressor/decompressor, `http://google.github.io/snappy/`, accessed September 16, 2023 (2015).

[19] C. Yann, lz4/lz4: Extremely fast compression algorithm, `https://github.com/lz4/lz4/`, accessed September 16, 2023 (2017).

[20] C. Yann, Zstandard - real-time data compression algorithm, `https://facebook.github.io/zstd/`, accessed September 16, 2023 (2017).

[21] J. Pope, A. Vafeas, A. Elsts, G. Oikonomou, R. Piechocki, I. Craddock, An accelerometer lossless compression algorithm and energy analysis for iot devices, in: 2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), 2018, pp. 396–401. doi:10.1109/WCNCW.2018.8368985.

[22] T. L. Le, M.-H. Vo, Lossless data compression algorithm to save energy in wireless sensor network, in: 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), IEEE, 2018, pp. 597–600.

[23] H. S. Mogahed, A. G. Yakunin, Development of a lossless data compression algorithm for multichannel environmental monitoring systems, in: 2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE), IEEE, 2018, pp. 483–486.

[24] D. Blalock, S. Madden, J. Guttag, Sprintz: Time series compression for the internet of things, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 2 (3) (2018) 1–23.

[25] D. A. Huffman, A method for the construction of minimum-redundancy codes, Proceedings of the IRE 40 (9) (1952) 1098–1101.

[26] T. Suel, Delta compression techniques, Encyclopedia of Big Data Technologies 63 (2019).

[27] S. Hardi, B. Angga, M. Lydia, I. Jaya, J. Tarigan, Comparative analysis run-length encoding algorithm and fibonacci code algorithm on image compression, in: Journal of Physics: Conference Series, Vol. 1235, IOP Publishing, 2019, p. 012107.

[28] Z. Zheng, Z. Zhang, A temporal convolutional recurrent autoencoder based framework for compressing time series data, Applied Soft Computing 147 (2023) 110797. doi:https://doi.org/10.1016/j.asoc.2023.110797.
URL `https://www.sciencedirect.com/science/article/pii/S1568494623008153`

[29] H. Feng, R. Ma, L. Yan, Z. Ma, Spatiotemporal prediction based on feature classification for multivariate floating-point time series lossy compression, Big Data Research 32 (2023) 100377. doi:https://doi.org/10.1016/j.bdr.2023.100377.
URL `https://www.sciencedirect.com/science/article/pii/S2214579623000102`

[30] Y. Mao, Y. Cui, T.-W. Kuo, C. J. Xue, Accelerating general-purpose lossless compression via simple and scalable parameterization, in: Proceedings of the 30th ACM International Conference on Multimedia, 2022, pp. 3205–3213.

[31] M. Goyal, K. Tatwawadi, S. Chandak, I. Ochoa, Dzip: Improved general-purpose loss less compression based on novel neural network modeling, in: 2021 Data Compression Conference (DCC), IEEE, 2021, pp. 153–162.

[32] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[33] I. H. Witten, R. M. Neal, J. G. Cleary, Arithmetic coding for data compression, Communications of the ACM 30 (6) (1987) 520–540.

[34] C. Zhang, Y. Miao, Q. Xie, Y. Guo, H. Du, X. Jia, Privacy-preserving deduplication of sensor compressed data in distributed fog computing, IEEE Transactions on Parallel and Distributed Systems 33 (12) (2022) 4176–4191. doi:10.1109/TPDS.2022.3179992.

[35] Y. Gao, L. Chen, J. Han, G. Wu, S. Liu, Similarity-based deduplication and secure auditing in iot decentralized storage, Journal of Systems Architecture 142 (2023) 102961. doi:https://doi.org/10.1016/j.sysarc.2023.102961.
URL `https://www.sciencedirect.com/science/article/pii/S1383762123001406`

[36] M. A. de Oliveira, A. M. da Rocha, F. E. Puntel, G. G. H. Cavalheiro, et al., Time series compression for iot: A systematic literature review, Wireless Communications and Mobile Computing 2023 (2023).

[37] J. D. A. Correa, A. S. R. Pinto, C. Montez, Lossy data compression for iot sensors: A review, Internet of Things 19 (2022) 100516.

[38] A. A. Sadri, A. M. Rahmani, M. Saberikamarposhti, M. Hosseinzadeh, Data reduction in fog computing and internet of things: A systematic literature survey, Internet of Things (2022) 100629.

[39] A. K. M. Al-Qurabat, C. Abou Jaoude, A. K. Idrees, Two tier data reduction technique for reducing data transmission in iot sensors, in: 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC), IEEE, 2019, pp. 168–173.

[40] A. K. M. Al-Qurabat, S. A. Abdulzahra, An overview of periodic wireless sensor networks to the internet of things, in: IOP Conference Series: Materials Science and Engineering, Vol. 928, IOP Publishing, 2020, p. 032055.

[41] W. B. Nedham, A. K. M. Al-Qurabat, An improved energy efficient clustering protocol for wireless sensor networks, in: 2022 International Conference for Natural and Applied Sciences (ICNAS), IEEE, 2022, pp. 23–28.

[42] S. A. Abdulzahra, A. K. M. Al-Qurabat, A. K. Idrees, Compression-based data reduction technique for iot sensor networks, Baghdad Science Journal 18 (1) (2021) 0184–0184.

[43] A. K. M. Al-Qurabat, S. A. Abdulzahra, A. K. Idrees, Two-level energy-efficient data reduction strategies based on sax-lzw and hierarchical clustering for minimizing the huge data conveyed on the internet of things networks, The Journal of Supercomputing (2022) 1–47.

[44] M. R. Chowdhury, S. Tripathi, S. De, Adaptive multivariate data compression in smart metering internet of things, IEEE Transactions on Industrial Informatics 17 (2) (2020) 1287–1297.

[45] Y. Xu, Y. Li, Q. Zhang, Z. Yang, Age-optimal hybrid temporal-spatial generalized deduplication and arq for satellite-integrated internet of things, IEEE Internet of Things Journal (2022).

[46] S. Salvatore, Redis is an in-memory database that persists on disk, `https://github.com/redis/redis/`, accessed September 16, 2023 (2009).

- Time-series data in the monitoring system remains a constant value for a continuous period.
- Utilize the continuous constant-value redundancy that exists in time series to compress data.
- Dynamic compression of data streams by dropping redundant data points on the IoT server side.
- Improve Iot-based monitoring system performance by dynamic data compression.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: