A novel compression approach for time series monitoring data

1st Carlos Rolo
Instaclustr
Netapp
Lisbon, Portugal
carlos.rolo@netapp.com

2nd Joshua Varghese
dept. name of organization
Open SI(NetApp)
Canberra, Australia
u3227463@uni.canberra.edu.au

3nd Ben Bromhead
Instaclustr
NetApp
Canberra, Australia
ben.bromhead@netapp.com

***Abstract*—This study introduces a novel approach to time series compression tailored for the complexities of computer systems monitoring. The Advanced Time Series Compression (ATSC) methodology, drawing inspiration from established audio compression techniques, achieves significant compression ratios. It enables storing raw data using 880 times less space, approximately an order of magnitude less than state-of-the-art compression methods. The data representation is converted from a sequence of points to a mathematical formula (and its parameters) by using dynamic block sizes to split time series and applying mathematical modeling to each block. Dynamic selection of the mathematical formula is based on information retrieved from the time series and allows an optimal fitting. The time component of the time series is reduced to a simple index that keeps the representation of time for each point and allows the precise retrieval and efficient streaming of data segments. ATSC presents a promising solution for effectively managing high sample rate time series data within the realm of computer systems monitoring. Here, we present promising results from initial testing against state-of-the-art compression systems.**

***Index Terms*—Time Series Compression, Function Approximation, Data Storage, Streaming, Indexing, Computer Systems Monitoring**

I. Introduction

In this study we introduce a novel method to compress time series monitoring data. Our approach enabled compression ratios of up to 3000x in test scenarios and up to 880x with production. Current monitoring data is lossy by nature, due to being a sampled, and as such, not capturing all events. And the sampling rate can have a varied rate, typically from 1 sample per second to a sample per minute [6], or even more [7]. Even so, capturing and storing monitoring data can take massive amounts of space. In our test case, a group of 57 nodes generated 600 MB in less than a day. Thus, monitoring data can be a significant burden in terms of storage and computational resources. Additionally, viewing data stored in the cloud incurs extra costs from egressing data. Current approaches tackle these problems by compressing time series data using several generic and/or specific algorithms [2] (e.g. Gorilla [8]). Domain-specific research for time series compression [5] [3] and improvements to existing algorithms [9] have been performed. Our approach leverages multimedia-specific knowledge (e.g., FLAC [10]) and the existing streaming capacity and fast decoding capabilities. Our approach starts with the premise that monitoring data is inherently lossy, and that we need the capacity to stream data in blocks to offer a fast decompression opposed to a slower compression process . We treat data as signals and leverage previously established signal processing techniques (e.g., Fast Fourier Transform (FFTs) and splines1) to approximate data points using mathematical formulas at the cost of precision. Using an approach that shares similarities with those used in domains such as audio compression, we developed a compressor capable of very high compression ratios, and fast decompression and streaming capabilities.

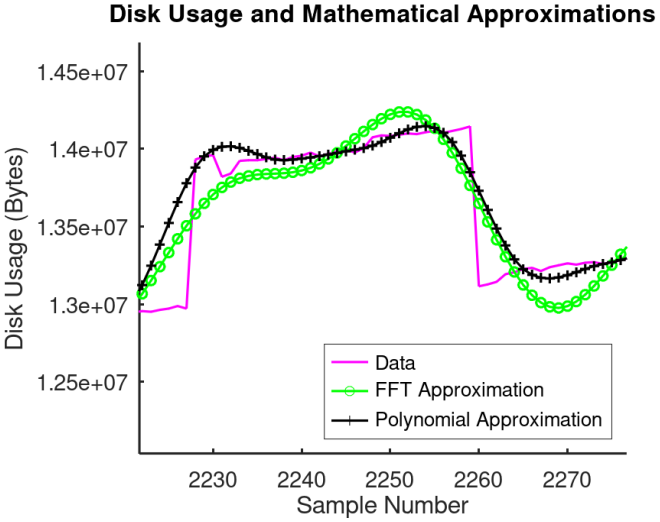

Fig. 1. A disk usage metric approximated by a Fast Fourier Transform and a Catmull-Rom interpolation.

Prior to the mathematical modelling of the signal, the data is split into blocks with a dynamic size. This is complemented by an indexing technique that allows the streaming of data without its complete decompression. The compressor can either operate in lossy or lossless mode. Importantly, even when operating in a lossy manner, the information is preserved . The compression ratios achieved using our method enables avoiding data aggregation techniques and culminate in storage and compute savings. In the following sections, we discuss the context leading to our research while exploring the architec-

ture of our compressor and providing a detailed explanation of its dynamic blocking, modelling, and indexing aspects. Furthermore, we expand on the topic of error correction for minimizing compression artifacts and discuss the results obtained with this approach on production datasets.

II. Background

In the dynamic landscape of computer systems monitoring, the impetus for our research stems from critical challenges that impact the efficiency and cost-effectiveness of current practices.

A. High Storage Cost

The exponential growth of data in computer systems monitoring has led to soaring storage costs. Traditional methods often struggle to cope with the sheer volume of information generated by processes, databases, and overall system health monitoring. Our research delves into innovative time-series compression techniques that alleviate the burden of high storage costs, offering a sustainable solution for data retention and management.

B. Balancing Data Reduction and Information Preservation

Conventional practices often resort to techniques such as averaging, sub-sampling or undersampling to manage the overwhelming data volumes. This approach effectively reduces the amount of data points but is undermined by the loss of valuable information. Our research addresses this trade-off by proposing advanced time-series compression methods that achieve data reduction without sacrificing critical insights and information4.

C. Computational Challenges in Traditional Monitoring Techniques

Traditional approaches, such as point-walking algorithms, demand substantial computational resources to process vast amounts of monitoring data. Recognizing the strain on computing capabilities, our method introduces a more resource-efficient paradigm by exploring novel compression techniques inspired by audio processing.

Our comprehensive approach tackles the immediate challenges of high storage and egress costs while addressing the inherent trade-offs of data reduction techniques. The overarching goal is to improve the landscape of computer systems monitoring by introducing a cost-effective and information-rich paradigm through advanced time-series compression.

III. Architecture

In this section we describe the internals of the ATSC compressor. That includes both compressor2 and decompressor, also internal structures and any other auxiliary mechanisms used.

A. ATSC Compressor

1) Split time and signal data: First part of the processing is to split the signal into is two components, time and the signal.

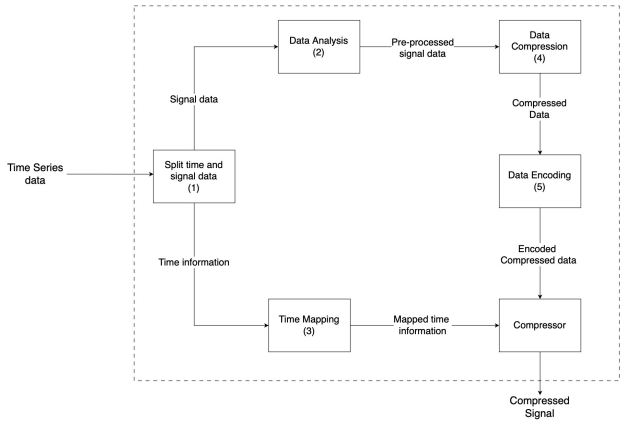

Fig. 2. Internal Diagram of the ATSC Compressor stage

2) Data Analysis: In this stage signal undergoes a statistical pre-processing, calculating the minimum, maximum, mean, variance, etc. Also, the splitting of the signal into different blocks is performed here. The block sizes are dynamic, and the sizing is done via 2 methods, either a discrete wavelet transform (DWT)3 or sizing for optimal FFT processing (in the form of $slice = 2^n * 3^m$). Given the computational cost of DWT, smaller signals are immediately sliced for FFT optimal size. In the other cases, DWT is performed first and if there is no high frequency components detected for slicing, FFT sizing is then performed. This information is provided to the Data Compression stage along with the signal data.

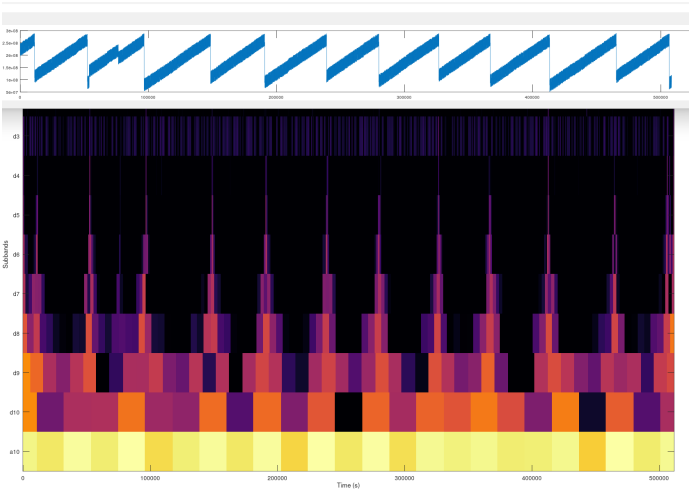

Fig. 3. Example of wavelet detection of high frequency sections to slice the signal. Signal on top and Wavelet Analysis on bottom.

B. Time Mapping

To facilitate precise access to samples without the need for full file seeking or decompression of the full file, ATSC generates a an index. This index ensures efficient retrieval of

relevant portions of the signal, a critical aspect in the decompression and streaming processes. The indexing technique is further explained in his own section.

1) Data Compression: The signal samples are divided into blocks, and for each block, ATSC applies modelling techniques tailored to the specific characteristics of the signal. This may involve techniques such as Linear Predictive Coding (LPC), Polynomial Prediction, Fast Fourier Transforms (FFT), or other methodologies that best suit the signal's behavior. For a special case of signals that are continuous only a single value is stored, alongside the number of samples. This modeling converts the original samples into an approximated formula. In some cases it is possible to match the original signal precisely and no other data is stored, otherwise an array of values representing the biggest outliers are stored alongside the formula type and its components. This leads to a high number of data points being converted into a mathematical formula and its components. Error correction and artifact reduction are also performed at this stage. A deeper dive into this is done in the CompressionIV section of this study.

2) Data Encoding: The last step of optimization is done by reducing the information to the minimal computational size needed for each sample. Samples are normally provided in 64bits size, this step can reduce the sample size to 8bit or any other size that the samples fit.

3) Compressor stage: This is the final stage, it just joins the compressed data alongside the index.

C. ATSC Decompressor

The decompression stage is fairly simple compared to the compressor. This allows a faster decompression, a prerequisite for a successful implementation. A decompression can be done for a full file, or for a specific segment of the file. The basic principle of operation is the following:

- ***Identifying the Samples to be Retrieved:*** Either all or a specific segment, the header of the file is read to find the block locations.
- ***Data Retrieval Using the Index:*** Check the index to identify in which blocks the samples are. In case of all the samples this step is ignored.
- ***Sample decompression:*** Once the relevant data segment is identified, ATSC initiates the decompression of the data. For the located data samples, the block information is retrieved, the correspondent formula and its parameters are used to construct the representation for the sample interval. Any error correction is also applied.
- ***Timestamp Integration Post-Decompression:*** Following decompression, the samples, now in their extracted form, receive timestamp information from the index. This final step ensures temporal accuracy and relevance of the retrieved data.
- ***Streaming/Writing data out:*** Under a streaming process, samples are sent while being decompressed. ATSC also supports full file decompression where all the data is decompressed and then send out, or written out to an output file.

IV. Compression Methods

Most of the work is done in this step. After a statistical analysis the signal, the compressor stage decides which approach is best for a given block of samples. Alternatively, a small subset of the block (128 samples) is compressed using all available compressors and the best one is selected. Depending on the error allowed for the compression, a less optimal method might be selected trading off precision for space savings. Each compressed block header always have the following information:

- Compressor used - 1 Byte,
- Minimum and Maximum value in the block - 4 Bytes each,
- Compressor parameters (e.g. FFT frequencies) - Variable,
- Error correction points - Variable;

1) Fast Fourier Transforms (FFT): Compression with FFT is done via a conversion of the signal to the frequency domain. Once in the frequency domain, we store a subset of frequencies. The number of frequencies selected depends on the error margin we want to store the signal with. Since the signals are always real, the worst case scenario (lossless compression) half the frequencies need to be stored. But since every frequency is represented by a pair of (Real, Imaginary) values, the lossless compression is next to none. The frequency pairs are always represented by two floats, either 32 or 64 bits depending on the precision required. In a lossy scenario, and depending on the signal, 10x reduction in size is easily achievable within a 1% to 3% error margin. To decompress, the Inverse FFT is applied on the available frequencies, zeroing out the missing frequencies.

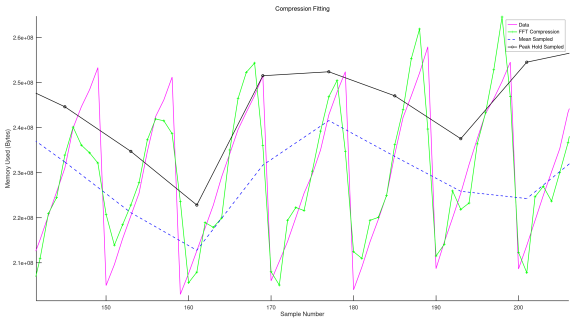

Fig. 4. Signal Fitting of the original data vs FFT Compression, Average Sampled and Peak Hold at the same compression ratio (8x).

2) Spline Interpolation: Another compression method is via a Spline Interpolation. For this study, we mixed Linear interpolation with Catmull-Rom interpolation, linear until we have enough data for a Catmull-Rom, and then we proceed with Catmull-Rom for the remaining of the signal. The method is similar to the FFT, where several points are stored, and the corresponding outputs calculated via Spline Interpolation. Once a Spline is generated within the margin of error provided, the points that generated that spline are stored. An advantage of Spline Interpolation vs FFT is that the FFT always need to

store the data as a pair of floats. While the Spline Interpolation the points are the same as the data points, so a integer signal would be stored as integer with the same bit depth as the original signal. This could be something as small as 8bit per point. In the decompression process, we do the interpolation calculation and retrieve the points requested. No need even to calculate the whole spline, only the segments corresponding to the samples.

3) Multivariate Interpolation: In an almost identical process to the Spline Interpolation, we used Inverse Distance Weighting as a method to approximate the signal. The results are solid, but almost the same as the spline interpolation, with the downside of a much slower decompression due to the complexity of the calculations. Since the approach is very similar, the results were very easy to compare, and in most of the cases the Spline Interpolation offers the same compression this method is used only in very rare cases. The similarity of the two also makes it difficult to choose each one to pick in a given scenario, and given the already mentioned disadvantages, this is only used in a manual selection of compressor methods.

4) Linear predictive coding (LPC): A lot of lessons were learned from initial experiments with Audio compressors (mostly FLAC encoder). Good results were obtained, and it was a natural approach to include the those within the available compressors. The approach is exactly the same as described in the FLAC format [11], but with a small change. The FLAC expects all samples to be integer, as such, floating point signals are discarded and not usable by definition. But for signals with a very small precision and bounded (e.g. 0.00 - 100.00) we can convert those to integer and use the LPC compressor. In practical testing, LPC offers the worst compression, worse than FFTs, so FFT is widely used, and in a similar case as the Inverse Distance Weighting, LPC is mostly relegated to a manual selection. The only case where LPC is used, is in lossless compression, where it offers compression averaging between 2x to 8x.

V. Artifact Detection and Correction

All the methods described above generate almost always, a lossy compression. But sometimes, the methods above might generate artifacts that remove quality, or might provide wrong information to a viewer of the signal5. Those artifacts are points that are generated from the mathematical approximations and are far from the original data points. They constitute point errors bigger than any other points in the generated data.

For reducing the error of the compression and/or remove artifacts from the generated signals the following process is used:

- Error detection via Mean Absolute Percentage;
- Storing of the errors that are above the defined error threshold;
- Store the location of the biggest errors and the correspondent original data point;
- When decompression happens, replace the generated points by the stored data.

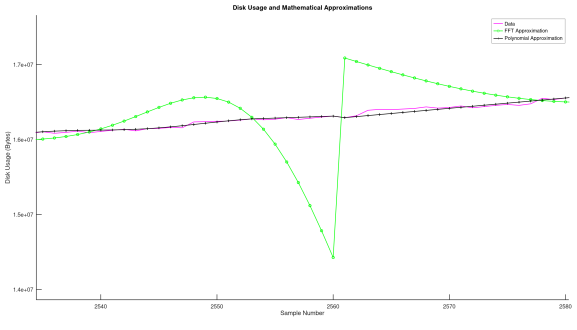

Fig. 5. Compression artifact from the FFT approximation.

This approach not only removes the biggest artifacts from the data, it also lowers the error rate of the signal. On a good fitting, we can avoid storing any point, so the importance of the fitting. A tradeoff exists, since a better fitting need more data stored (e.g. More frequencies for FFT), and storing errors need space. For lossless compression all the point differences need to be stored. In a lot of cases this leads to almost no compression unless the signal has a perfect fitting with the mathematical representation.

VI. Indexing Samples with VRSI (Variable Rate Sampling Interval)

1) Overview of VRSI Mechanism: In the methodology section, a critical aspect of the process involves indexing samples for efficient data retrieval. Variable Rate Sampling Interval (VRSI) is employed to manage the sampling intervals effectively. VRSI operates on the premise that samples occur at evenly spaced intervals, such as every 5 seconds, 20 seconds, or 60 seconds.

2) Line Segment Representation: For each sampling interval, a line segment is created with specific fields:

- **Start Timestamp:** The timestamp marking the beginning of the sampling interval.
- **Sample Interval:** The time gap between consecutive samples.
- **Starting Sample:** The index of the first sample within the segment.
- **Number of Samples:** The total number of samples within the segment.

3) File Structure: These line segments are then stored in a file, accompanied by the lowest and highest timestamps in the segments represented. This file structure allows for easy identification of whether a requested interval is present in the index. If the requested interval falls entirely outside the timestamps in the file header, no samples are available for that interval.

4) Handling Temporal Gaps: In cases where a metric stops reporting for a period, a new line segment is generated, ensuring accurate representation of the sampled data over time.

5) Example VRSI File Content:

```
55745
59435
15,0,55745,166
15,166,58505,63
```

- Line 1) Represents the first timestamp.
- Line 2) Represents the last timestamp.
- Line segments are detailed below:
  - The first line segment has one sample every 15 seconds, starting at timestamp 55745, with a total of 166 samples.
  - The second line segment also has one sample every 15 seconds, starting at timestamp 58505, with 63 samples.

6) Locating Samples (Read Path): In the read path, when locating a sample in a file using the index, timestamps or timestamp ranges are specified (e.g., "All the samples from 3:30 PM to 4:30 PM"). The system checks if the requested timestamps are within the available timestamps in the file header. If found, the sequence of sample numbers is extracted, indicating the samples needed for decompression.

7) Creating/Updating the Index (Write Path): When a new sample is added, the index is updated. Since time progresses linearly, and samples occur in sequence, the system only needs to:

- Update the last segment's sample number or
- Create a new segment.

Existing segments are incremented if the sample timestamp is the next in sequence. If a segment does not exist or the timestamp is not the next in sequence for the latest segment, a new segment is created. This approach ensures an efficient and organized index for managing variable rate sampling intervals.

VII. Results

A comprehensive testing methodology was employed to assess the efficiency and effectiveness of ATSC. The ATSC server was configured as both a read and write backend for a Prometheus [1] instance, establishing a practical and relevant testing environment. In turn, the Prometheus instance was connected to an Instaclustr internal production 57-node Cassandra [4] cluster with a Prometheus endpoint enabled. The data flow was then collected at the node level and sent to the Prometheus endpoint that would forward it to the ATSC server configured as previously stated. The server ran for approximated 18 hours over a 2 day period. Samples were collected at 1 point every 20 seconds, resulting in 5432 samples per each signal for each node. A total of 14386 signals were processed. We removed signals representing aggregations (e.g. histograms), reducing the total number of signal to 13950. ATSC ran with automatic compressor selection and a maximum allowed error of 3%.

A. A Best-Case Scenario Analysis

Before delving into the detailed results, it is essential to address the ATSC single scenario. The best case scenario for ATSC is a dataset that fits perfectly with a mathematical model. In such a case, even with a growing number of samples it is possible to store a massive amount of data represented by only the mathematical formula, the sample number, and the index. In these circumstances, compression ratios above 3000x were achieved. This ideal scenario is also realistic, as stale or infrequently changing signals always exist. In our production tests, approximately 13% of the data falls into these categories.

Method	Compressed size (bytes)	Compression Ratio
Prometheus	454,778,552	1.86
LZ4	141,347,821	5.99
ATSC	14,276,544	59.35

TABLE I
BENCHMARK RESULTS OF COMPRESSION PERFORMANCES BETWEEN DIFFERENT COMPRESSION METHODS.

B. Data Overview

Raw Data Size: 847,315,029 bytes

Compression Statistics:

The compression performance of ATSC was benchmarked against two state-of-the-art methods (Prometheus and LZ4). results obtained in the compression tests are shown in Table 1 and Figure 7. Overall, ATSC performed substantially better, achieving a compression ratio over an order of magnitude than the other methods tested (Table 1). The data overview in Table 1 provides a performance reference, while compression statistics and signal counts measure the efficiency of ATSC's compression strategies. The compression effectiveness was quantified using the average compression ratios and the compression histogram6.

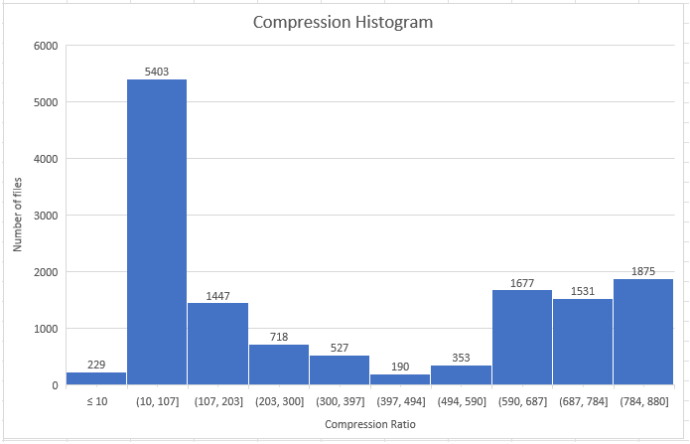

Fig. 6. ATSC Compression Ratio Histogram

C. Validation of results

The previous results were validated by uncompressing the compressed data and comparing it with the original signal. The compressor chose to perform lossy compression, which was expected because it was set to automatically determine the best compression strategy and given a maximum allowed error.

The degree of information loss varied for different signals but the overall performance was remarkable. For example, the mathematical approximations of "CPU usage" (Figure 7) and "Disk usage" (Figure 8) were very close to the original data. The latter resulted in an error of less than 3%. "Heap usage" is a signal that is normally very difficult to fit. Despite this, ATSC tracked the signal with an accuracy that was sufficient to capture the information. As stated before, reducing the compression ratio can lead to a better approximation. However, our tests on real data shows that this difference is not significant, this is demonstrated in figure 9.

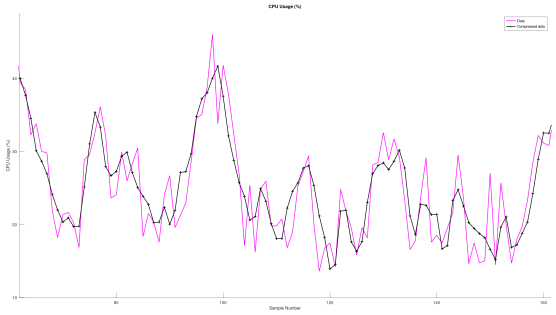

Fig. 7. CPU compression (17x) vs Original data

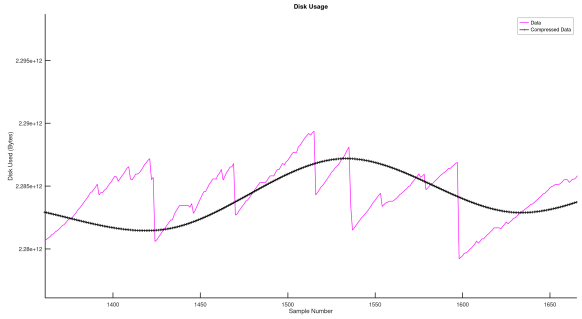

Fig. 8. Disk compression (146x) vs Original data

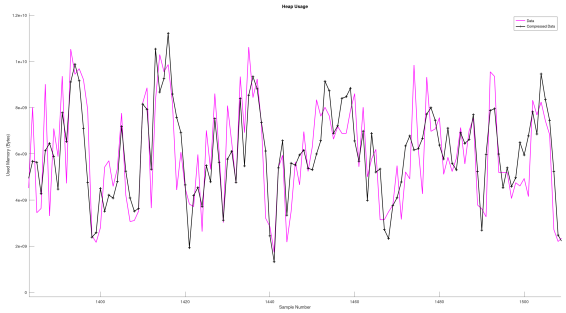

Fig. 9. Heap Usage Compressed (18x) vs Original data

In conclusion, ATSC compression enabled compression ratios of approximately 60 times, achieving remarkable data reduction. These results substantiate ATSC's capabilities in optimizing storage space, offering valuable insights into its potential impact on real-world applications.

VIII. Future Work

While ATSC shows a significant improvement over existing compression techniques for time-series, several work directions can further expand its compression ratios, system integration, and others.

A. Compressor selection and optimization

Currently, ATSC provides a small subset of fixed algorithms. The choice is made based on a simplistic statistical analysis. In the future, choosing the optimal algorithm for different datasets could be performed by an Artificial Intelligence (AI) model trained using a large number of signals. Or more robust and/or faster methods for

Test further algorithms

ATSC algorithm selection can be expanded alongside improved selection methods to further increase its achievable compression ratios.

B. Integration with other systems

While developing ATSC, we found that it creates similar outputs for many signals. This knowledge can be leveraged by having ATSC integrated in a database to identify such scenarios and avoid storing data once a specific output already exists. This would substantially improve space savings.

C. Outlier Detection

During our tests, we noted that the compression ratios for a given metric across all nodes would be mostly the same for systems with several nodes doing the same work (e.g., a Cassandra database with multiple nodes). In some cases, we found that nodes had significant differences for some metrics, representing 2 to 10 times worse compression. na in-depth look into the detailed metrics showed that the node was misbehaving, having to be fixed or replaced. This ability to detect outliers by comparing compression is an application worth exploring.

D. Use of Hardware Capacities

Our approach yields savings in at-rest scenarios, as well as in transit and during compute processing. The ability to leverage client-side devices [9] enables efficient processing and transport of monitoring data. Integration in system-on-a-Chip architectures [8] could lead to further performance enhancements. This strategy positions our methodology at the forefront of efficient and resource-conscious computer systems monitoring.

IX. Conclusion

In conclusion, the ATSC approach outlined in this study offers a novel solution for monitoring time-series data compression, particularly in computer systems. ATSC achieves impressive compression ratios, achieving over 1000x times compression. This innovative methodology addresses current storage and data transfer challenges. Preliminary tests against a production cluster demonstrated significant space savings using ATSC compared to current state-of-the-art methods. This study emphasizes the importance of unique indexing for precise data retrieval, showcasing its efficiency in streaming and targeted decompression of relevant data segments.

The future work directions outlined underscore ATSC's commitment to continuous improvement, with a focus on automated selections, integration with time-series databases, and addition other capabilities. These efforts aim to keep ATSC at the forefront of efficient and intelligent time-series data compression.

References

[1] Prometheus Authors. Prometheus - from metrics to insights, 2024. [Online; accessed 1-March-2024].

[2] Hongze Cheng. Compressing time series data. Technical report, TDEngine, October 2022.

[3] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. A time-series compression technique and its application to the smart grid. *The VLDB Journal*, 24:193–218, 2015.

[4] The Apache Software Foundation. Apache cassandra - open source nosql database, 2024. [Online; accessed 1-March-2024].

[5] S. Edward Hawkins III and Edward Hugo Darlington. Algorithm for compressing time-series data. Technical report, Johns Hopkins University Applied Physics Laboratory for Goddard Space Flight Center, June 2012.

[6] Alexis Lê-Quôc. Monitoring 101: Collecting the right data. Technical report, Datadog, June 2015.

[7] n. a. Azure monitor metrics aggregation and display explained. Technical report, Microsoft, June 2023.

[8] J. Teller et al. Gorilla: A fast, scalable, in-memory time series database. *Proceedings of the VLDB Endowment*, 8(12):1816–1827, 2015.

[9] Aliaksandr Valialkin. Achieving better compression for time series data than gorilla. Technical report, VictoriaMetrics, May 2019.

[10] Martijn van Beurden and Andrew Weaver. Free lossless audio codec. Technical report, IETF, June 2017.