Adaptively Compressing IoT Data on the Resource-constrained Edge

Tao Lu
Marvell Tech. Group

Wen Xia,* Xiangyu Zou
Harbin Institute of Technology, Shenzhen, China

Qianbin Xia
Marvell Tech. Group

Abstract

Big IoT data needs to be frequently moved between edge and cloud for efficient analysis and storage. Data movement is costly in low-bandwidth wide area network environments. Data compression can dramatically reduce data size to mitigate the bandwidth bottleneck. However, compression is compute-intensive and compression throughput can be limited by available CPU resources. The impact of available computation capability of the resource-constrained edge on the edge-to-cloud data transfer rate is apparent. Our study reveals compressors, including *gzip*, *bzip2*, *lzma*, and *zstd*, perform very differently under various resource-constrained conditions. This motivates us to propose models for the best compressor selection under CPU, network, and storage resource limitation conditions on the edge. We implement *ZipMate*, a middleware that enables resource-aware and adaptive compression policy based on the model. Our evaluation shows that adaptive policies consistently outperform unitary or random compressor selection policies.

1 Introduction

We are living amid an invisible revolution driven by the Internet of Things (IoT). IoT consists of billions of connected devices that are transforming many domains, continually producing data. IDC suggests that by 2025, there will be 41.6 billion connected IoT devices or things, generating more than 79 zettabytes (ZB) of data [5]. According to an NSF Data Storage Research Report [10], IoT brings an explosion of data collection, storage, and processing demands. It is crucial to identify and reduce IoT data in a timely fashion, as well as balancing among storage, preprocessing, and communication between the resource-constrained IoT devices and cloud.

The lifecycle of data within an IoT system proceeds from data production at IoT devices to data aggregation, transferring, filtering, preprocessing, preliminary analysis and consumption, and finally in-depth analysis, permanent storage and archiving. Data collection, aggregation, simple querying and even preliminary processing can be conducted at the edge. Limited by the available computing power and storage space at the edge, IoT data are usually moved from the edge to the cloud for in-depth analysis, long-term storage, and permanent archiving.

Moving data from the resource-constrained edge to the cloud is costly [21,24,29,30]. Zhang et al. [30] propose Global Data Plane to change cloud-centric IoT architecture into data-centric to avoid the pitfalls. Bharde et al. [12] observe the similarity of time series data, and they apply discrete wavelet transform based deduplication to considerably reduce the size of video frames data needs be transferred in autonomous car environments. Gupta et al. [18] recognized that edge-cloud datastore has data items being partitioned across edge and cloud nodes, causing detrimental delay of queries. They propose a simple compression approach to reduce the time series data to reduce edge-to-cloud data access latency. Altarawneh et al. [9] propose only storing the extracted result data and metadata for video streaming applications, instead of sending full data from the edge to the cloud. Elgazar et al. [17] propose smart media compression, which takes file popularity and dynamic networking environment into consideration to reduce the file access latency. Chen et al. [14] propose to compress sensor data to improve real-time performance of the industrial robot system. Stojkoska et al. [27] apply delta compression to temporally correlated data for energy saving. Azar et al. [11] use *SZ* [16] to compress wearable device data on an Android device, reducing the Android to edge traffic by up to 103 times. Taking limited processing capability of low-power IoT devices into consideration, Blalock et al. [13] introduce a time series compression algorithm called Sprintz, which achieves high compression ratios while adding little memory and latency overheads. In the traditional file and volume storage scenario, Harnik et al. [19] propose prefix-based heuristic estimation to predict the effectiveness of *zlib* in real time so as to make the 'compress or not' decision.

Existing IoT data reduction each mainly targets a specific application scenario. There lacks a universal framework that cohesively considers multiple optional compressors, data heterogeneity, and available server and network resource constraints to make the best compressor selection on IoT edges. To fill this gap, we are conducting the following work. First, we study datasets, obtaining important observations about data compressibility, compressor performance on the datasets, and the potential benefits of applying specific compressors to the data. Second, we build mathematical models to recognize data compression or transfer bottlenecks under CPU, network, and storage resource limitations, to make optimal compression policy for a dataset in real time before transfer. Third, we implement *ZipMate*, a middleware for efficient edge cloud storage management. *ZipMate* is under development, and the version presented in this paper is archived at GitHub/ZipMate. We present preliminary results of our adaptive compression.

*Corresponding author: xiawen@hit.edu.cn.

Table 1: Data compressibility evaluation using four general compressors *gzip*, *bzip2*, *lzma*, and *zstd*, as well as specific (denoted as *spec*) compressors including *webp* for images (i.e., Data 3,6), *SZ* for floating-point/integer data (i.e., Data 1,5,7,9), and *gdcmconv* for DICOM medical images (i.e., Data 2).

Data	IoT Domain / Scenario / Device	Format/Type	Compression Ratio | Throughput (MB/s)				
			gzip	bzip2	lzma	zstd	spec
1	Energy/Appliance Power/ Meter [25]	CSV/Integer	24.1 | 3.2	30.1 | 4.7	28.2 | 4.8	23.8 | 10.5	50.0 | 148
2	Health Care/ Cancer/ DCE-MRI [15]	DCM/Image	2.4 | 6.8	3.3 | 4.8	2.8 | 7.8	2.5 | 10.0	3.1 | 13.6
3	Space Science/Exoplanet/Satellite [6]	TIFF/Image	1.4 | 21.9	1.86 | 12.2	1.82 | 2.5	1.49 | 10.9	2.6 | 4.0
4	Petro Science / Oil Well / Meter [28]	CSV/Float	8.0 | 6.8	10.4 | 11.9	15.8 | 2.0	9.6 | 15.9	N/A
5	Agriculture / Soil / Sensor [26]	CSV/Float	2.6 | 3.2	2.9 | 9.8	2.6 | 6.4	2.6 | 5.2	10.0 | 34.0
6	Biology / Cell / Simulator [22]	TIFF/Image	1.8 | 3.9	2.3 | 8.2	2.0 | 5.9	1.9 | 8.7	2.7 | 0.9
7	Climate / Weather / Thermometer [1]	CSV/Float	3.1 | 5.8	3.8 | 2.9	3.0 | 2.3	3.2 | 2.3	3.1 | 6.1
8	Smart City / User Study / Phone [3]	JSON/K-V	6.3 | 19.7	7.5 | 7.6	8.6 | 18.4	12.6 | 22.1	N/A
9	Earth / Ocean / Sensor [4]	CSV/Float	2.8 | 5.0	3.2 | 9.6	2.9 | 7.0	2.9 | 10.7	3.9 | 14.5

2 IoT Datasets Study

We study multiple IoT datasets from the *Awesome Public Datasets* repository [2], which contains hundreds of datasets crossing tens of application domains. As Table 1 shows, some datasets are collected from typical IoT devices, such as smart farming application sensors which record soil moisture data. Some datasets are not claimed to be collected by IoT devices, but their application scenarios could be typical or emerging IoT scenarios. For example, we adopt satellite images because connecting IoT devices through satellites is emerging. We conduct data compression on data samples with *gzip*, *bzip2*, *lzma*, and *zstd* (the experimental setup is presented in Section 4.1). Some specific compressors on corresponding data types are also studied, such as *SZ* [16] on floating-point and integer data, *webp* [8] on TIFF image, and *gdcmconv* [7] on DCM medical images. About the sampling method, for small datasets we simply use the whole data, for large datasets, we use the first file in the dataset folder. The samples and test scripts are included in the codebase we have shared. We make the following key observations in the dataset study.

First, IoT data exhibits good compressibility. For most datasets, general compressors can achieve a compression ratio higher than 3×, some even over 10×. The *gzip* compressor, which usually delivers the lowest compression ratio, can even reduce the data size by over 60% on average. Thus, we envision data compression in the edge cloud path can considerably reduce data transfer traffic, and shorten data access latency.

Second, no single compressor can be fit for all datasets. We observed that *bzip2*, *zstd*, and specific compressor *SZ*, *webp*, and *gdcmconv* are winners on different datasets, respectively. There is no single compressor that can win on all datasets or most datasets. In contrast, on different datasets, compressors demonstrate different compression ratio or rate. In heterogenous data environments and changeable computation resource conditions, adaptive compressor selection can be very beneficial.

Third, specific compressors can dramatically outperform general compressors in either compression ratio, or rate, or even both. Floating-point data and image data are pretty common, especially in IoT scientific application scenarios. *SZ* was a lossy compressor originally designed for floating-point data in HPC environments. Our tests show that *SZ* dramatically outperforms general compressors on both compression ratio and throughput (about 2× and 20× higher respectively), only at the small cost of an error-bounded fidelity loss. Screening and integrating specific compressors in *ZipMate* and intelligently using them is our ongoing work.

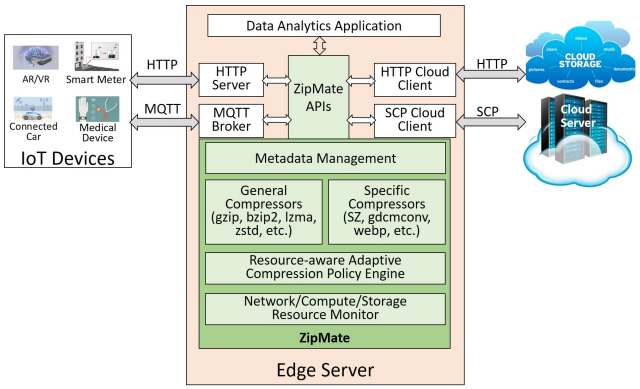


Figure 1: A system overview of *ZipMate* on the edge.

3 Adaptive Compression Design

Figure 1 shows the operating environment of *ZipMate*. We assume various IoT devices such as camera devices, connected cars, smart sensors, telemetric devices, medical devices, etc., collect source data, and then send them to edge servers for storage, processing, and analysis. When the IoT data accumulate at the edge, becoming big data, requiring in-depth analysis or permanent storage, they are pushed to the cloud cores. Our adaptive compression aims to minimize data transfer time from the edge to the cloud or maximize the compression ratio, depending on user-defined requirements.

To make the middleware portable, we believe it should

have minimal overlap with the 'device-to-edge' and 'edge-to-cloud' communication protocols in the system I/O path. In our current design, *ZipMate* provides APIs to HTTP servers and MQTT brokers to write data received from IoT devices into edge local storage. *ZipMate* also provides APIs to HTTP or SCP cloud clients to retrieve edge local data and send it to the cloud for analysis. Currently, *ZipMate* implements *put, get, list, and delete* APIs for storage management:

The job type, that's the purpose of edge-to-cloud data movement, which we suppose to be user-defined, is important to make proper compression policy. We target two representative data movement scenarios. In the first scenario, data needs to be moved from edge to the cloud for in-depth analysis. We call this scenario *computation offloading*. In this case, we treat the edge-to-cloud data transfer a business critical and time-sensitive task. The data transfer should be finished as soon as possible, so analysis can quickly start, insights and knowledge can be obtained in time. In the second scenario, data transfer to the cloud is not for analysis, but for releasing edge storage capacity. We treat this task time-insensitive, thus can tolerate long transfer latency. Instead, storage space saving is the most important metric, which demands the highest data compression ratio. We call this scenario *storage offloading*.

We are discussing our adaptive compression design on the two scenarios respectively, using mathematical models to recognize data compression or transfer bottlenecks under CPU, network, and storage resource limitations.

Scenario 1: For *computation offloading* jobs, adaptive compression aims to accelerate the data transfer so that analysis of IoT data in the cloud can be launched as soon as possible.

In *Scenario* 1, data needs to be transferred from the edge to cloud servers as soon as possible, and the best compressor should be the one that enables the fastest data transfer, in terms of uncompressed data size. Symbols are summarized in Table 2. For a specific compressor c, assuming its compression throughput on a dataset is $T(c)$ with computation power of one CPU core, and the available spare utilization of i_{th} CPU core for compression is A_i, then the potential aggregate compression throughput $T_A(C)$ of all cores can be calculated as:

$$T_A(c) = T(c) * \sum_{i=1}^{n} A_i \tag{1}$$

Data compression needs to read data from storage systems, thus the storage system's available I/O bandwidth also limits the achievable compression rate. Assuming the available I/O bandwidth is D, we get the aggregate compression throughput with storage system I/O limit:

$$T_L(c) = min(T_A(c), D) \tag{2}$$

Assuming the compression ratio of compressor c is $R(c)$, and the available network bandwidth is B, which is limited by both available edge network bandwidth and cloud storage bandwidth, then the data transfer throughput, in terms of uncompressed size, that can be supported by the network is $B * R(c)$. Cloud-side decompression can be conducted in parallel with edge-side compression, and decompression is much ($\sim 4\times$ for gzip) faster than compression, thus we do not specifically consider decompression in the end-to-end transfer time. Taking run-time available CPU computation power, storage I/O bandwidth, and edge-to-cloud network bandwidth into consideration, the data transfer throughput is:

$$T_N(c) = min(T_L(c), B * R(c)) \tag{3}$$

In *Scenario* 1, the best compressor c is the one that can deliver maximal $T_N(c)$. To screen out the best compressor, there are a few challenges.

First, it is challenging to predict compression throughput $T(c)$ and compression ratio $R(c)$. For a specific compressor c, $T(c)$ and $R(c)$ are both dataset dependent. They can not be known a priori. Considering the computation cost, it is not feasible to compress the full dataset to get $T(c)$ and $R(c)$. We need mechanisms to accurately and efficiently predict $T(c)$ and $R(c)$ for a specific dataset. Our previous work [23] shows that the accuracy of simply using data features such as byte entropy and coreset size to predict the compression ratio is not satisfactory. We have observed that there is a more direct relationship between compressor data structures for encoding and compression ratio. Therefore, we propose to sample a small subset of a full dataset, conducting compression on the sample to get the performance data and compressor encoding data structures information, to predict $T(c)$ and $R(c)$. To achieve the best prediction accuracy, sampling methods should be compressor specific and compressor internal encoding data structures need to be probed. To integrate tens of compressors in *ZipMate*, investigating sampling methods and building prediction models are real challenges.

Second, it's challenging to monitor and accurately calculate available computation power, storage I/O, and network bandwidth in real-time. Calculating computation power and network bandwidth utilization are relatively straightforward. It's very challenging to calculate available storage I/O bandwidth, because storage system performance is workload dependent while the sequential I/O and random I/O perform

Table 2: Symbols used in Models for adaptive compression.

Symbol	Description
A_i	Available/Spare CPU utilization of $Core_i$ (1 <= i <= N).
B	Available edge-to-cloud network bandwidth.
c	Representing a compressor.
D	Available storage system I/O bandwidth.
N	Number of CPU cores.
$R(c)$	Data compression ratio of a compressor.
$T(c)$	Compression rate of a compressor with one dedicated core.
$T_A(c)$	Compression rate under CPU limitation.
$T_L(c)$	Compression rate under CPU and storage limitation.
$T_N(c)$	Compression rate under CPU, storage, network limitation.

very differently. Moreover, we expect storage systems to be shared by multiple workloads and I/O interference a real problem. We plan to measure the storage system pressure from two dimensions, bandwidth for sequential I/O and IOPS for random I/O, to achieve a reasonable calculation.

Scenario 2: For *storage offloading* job, adaptive compression aims to reduce data as much as possible, maximizing cloud storage cost savings.

In *Scenario* 2, a data transfer job intends to release storage capacity of edge storage servers. In this case, data needs to be compressed as much as possible to minimize the data volume and maximize cloud storage cost savings. This kind of job can be categorized into data archiving, which does not have strict timing requirements and can be treated as a job with low priority, being processed when the computation resources are sufficient. Therefore, **the best compressor c for *Scenario* 2 is the one that achieves the highest $R(c)$**. Meanwhile, if the challenges in *Scenario* 1 are well resolved, selecting the best compressor in *Scenario* 2 will be straightforward.

Metadata Module. The metadata fields of file objects in *ZipMate* are stored at edge servers. This is different from cloud storage where metadata are stored along with objects because *ZipMate* file objects are not plain files but compressed or encrypted, and the decompression or decryption operations are conducted at the edge which require the metadata information. When file objects are at large scale, we expect metadata management to be a big challenge. Sophisticated metadata management is our on-going work.

4 Prototype and Evaluation

4.1 Experimental Setup

We use an x86 workstation as the edge server, which is deployed with a 4-core Intel(R) Core(TM) i5-3470 CPU @ 3.2GHz, totally 16 GB DRAM running a Linux Ubuntu Server (x86_64). It communicates with a Google Computer Engine server with a machine type *e2-standard-2* (2 vCPUs, 8 GB memory), located at *us-west1-a* zone, which serves as the cloud server. *ZipMate* runs on the edge server, supporting two protocols: *scp* when the cloud storage is backed by Google Compute Engine Server and *http* when the cloud storage is backed by Google Drive. Our evaluation is based on *scp* protocol. We use software at edge servers to accurately control the edge upload network bandwidth, and set different compression threads in *ZipMate* to control the CPU cores being used, so as to emulate various resource conditions. In our tests, we do not limit storage system bandwidth and thus it's not a real bottleneck in any test case.

We use the first 4 datasets in Table 1 for evaluation. Specifically, Data 1, 2, 3, 4 consists of 5, 12, 6, 9 and files, totally 227, 126, 123, and 192 MB, respectively. We treat the 4 datasets as an ensemble, 668 MB in total, using edge-to-cloud transfer time of the ensemble as the metric to measure the performance of compressors. For adaptive compression we need samples to predict data compression ratio and rate. For Data 1, 2, and 4, we employ prefix sampling [19], which simply takes the first 1 MB contents of the data as samples. For Data 3, which consists of large *tiff* images, we notice prefix sampling is not accurate. Therefore, we simply use a small 3MB *tiff* file as the sample. We configure compressors with default parameters. The only exception is *lzma*, because its default mode is too slow. We set the *lzma* speed parameter as '-1', indicating the fastest mode to make it competitive.

4.2 Results

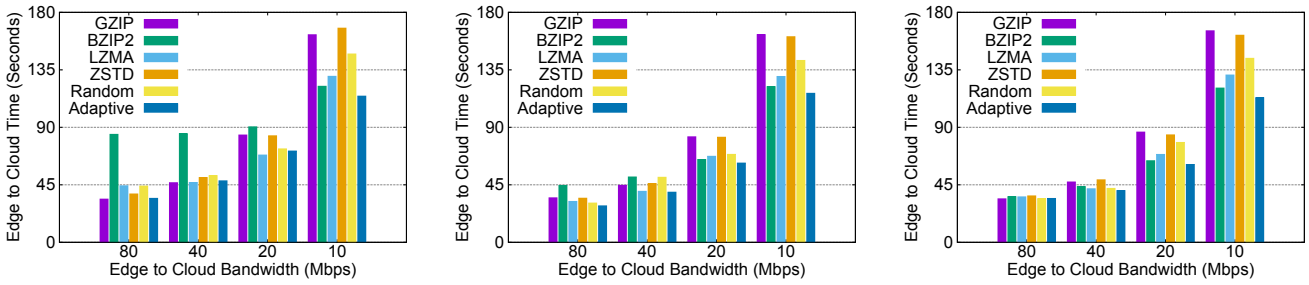
In this subsection, we discuss our key observations from the evaluation of adaptive compression.

Observation 1: Network bandwidth and computing power both have obvious impact on data transfer time, but the bandwidth dominates the impact.

As Figure 2 shows, in general the edge-to-cloud data transfer time decreases when network bandwidth or the number of CPU cores increases. For example, with 1 core for compression, when network bandwidth increases from 10 to 20, 40, and 80 Mbps, the average data transfer time dramatically reduces from 145 to 81, 57, and 50 seconds respectively. With 2 or 4 cores for compression, increasing network bandwidth has a similar impact on data transfer time. In comparison, the impact of increasing CPU cores for compression is not as much as increasing network bandwidth. At 80 Mbps network bandwidth, as the number of CPU cores increases from 1 to 2 and 4, the edge-to-cloud transfer time with *bzip2* dramatically reduces from 84 to 44 and 36 seconds. However, for *zstd*, the data transfer time only slightly changes from 38 to 35 and 36 seconds, respectively. The reason is that the default speed mode of *bzip2* is more compute-intensive than *zstd* and other compressors. As a result, with only one core, compute capability is the main bottleneck of *bzip2*. As the number of cores increases to two, network bandwidth bottleneck emerges for *bzip2*, and dominates for other compressors. Thus, further increasing the number of CPU cores to four does not reduce the edge-to-cloud data transfer time much.

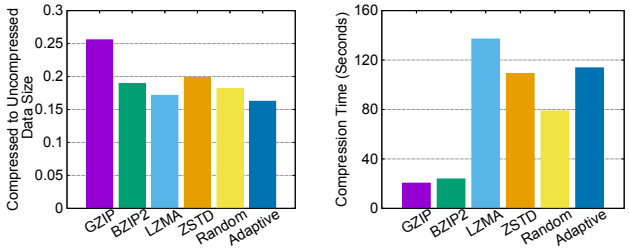
Observation 2: There is no single compressor which performs best in all or even most cases, either in terms of edge-to-cloud transfer time in resource limitation conditions, or in terms of highest compression ratio.

Compression introduces overheads, if compression does not bring benefits, no compression is also an option in our framework. For the datasets we have tested, all compressors can dramatically reduce data size, shortening data transfer time especially in low-bandwidth conditions. For example, when the bandwidth is 10Mbps, even the worst *gzip* can reduce data transfer time by 74%, compared with transferring uncompressed data. Our preliminary evaluation on datasets listed in Table 1 has shown that compressor performance is dataset dependent. Our comprehensive tests further validated



(a) 1 Core (b) 2 Cores (c) 4 Cores

Figure 2: Edge-to-cloud end-to-end data transfer time integrating various data compression schemes with 1, 2, 4 CPU cores under various network bandwidth. Compressors run at their default configurations, except LZMA, which is reconfigured to its fastest mode, because the default mode is too slow to be competitive.



(a) Compression Impact (b) Compression Time

Figure 3: Extreme compression ratio and time cost comparison at best compression ratio mode with 4 cores.

this observation. When the network bandwidth is 80Mbps, compressing with 1 core, *gzip* is the best compressor. However, as the network bandwidth reduces to 10Mbps, *gzip* becomes the worst. Also, with 40Mbps bandwidth and 1 core, *bzip2* is the worst. However, with 4 cores and the same bandwidth, *bzip2* becomes the best.

> ***Observation 3***: The resource-aware adaptive compressor selection scheme consistently outperforms unitary or random solutions in either achieving the shortest edge-to-cloud transfer time or achieving the highest compression ratio.

We compare adaptive compression with unitary and random compressor selection. We implement multi-threading to consume 1, 2, and 4 cores for compression to emulate various computation resource limit conditions. We limit network bandwidth to 10, 20, 40, and 80 Mbps to emulate various network resource limit conditions. We always use 4 threads for data transfer to ensure network bandwidth is maximally utilized. Our evaluation shows that in 9 out of the 12 test cases, our adaptive compression outperforms the best unitary compressor. In the other 3 cases the adaptive compression is only outperformed by 1%, 3%, and 4.5%, respectively. On average, our adaptive compression outperforms unitary compressor selection by 12% to 31% in 1-core condition, by 14% to 22% in 2-core condition, and by 10% to 22% in 4-core condition. Averaging all 12 test cases, adaptive compression is 13% faster than random compressor selection.

In *Scenario* 2, compression ratio is the most significant metric. Therefore, we compare compressors running at their best compression ratio mode. As it shows in Figure 3, for the 4 datasets we tested as an ensemble, *lzma* and *bzip2* perform better than *zstd* and *gzip* in compression ratio. Our adaptive compression scheme can choose the better one between *lzma* and *bzip2*, thus achieving a better compression ratio than any unitary compressor or random compressor selection. Specifically, *adaptive* outperforms the best unitary compressor *lzma* by 5% in compression ratio with 17% less compression time.

5 Conclusion and Future Work

For a dataset, different compressors show contrasting performance. In resource limitation conditions, compressors get stuck due to various resource bottlenecks, which has motivated us to propose adaptive schemes for resource-aware compressor selection. Our preliminary evaluation validates the effectiveness of adaptive compression.

We plan to extend our work in four directions. ①, we continue enhancing our adaptive compression model, especially how to properly sample a dataset so as to make accurate compression ratio and rate prediction, which we believe is very challenging but crucial to our model accuracy. ②, we plan to compare the data feature-based black-box method with our compression encoding based white-box method. Jin et al. [20] recently proposed using Long short-term memory (LSTM), a deep learning method to extract data features and make adaptive compression algorithm selection for database storage. The potential of using these sophisticated models for compression ratio prediction in our scenario is unclear and we plan to explore. ③, as data compression and transfer cause computation, storage, and network resource competition, we plan to run representative edge analytics workloads to measure the performance degradation of other edge applications caused by compression and propose a solution to mitigate the performance degradation of the business-critical edge tasks. ④, we understand the importance of data security and privacy. We plan to comprehensively investigate the security challenges in IoT, edge, and cloud environments, to enforce proper data security policies in our middleware.

6 Discussion

Feedback expectations: A bunch of related work supports our observation that data transfer between edge and cloud dramatically stresses the network link. Our preliminary tests show that data compression is effective to mitigate the network link bottleneck. We especially expect two kinds of feedback. First, why there has not been a solid and universal data compression solution in this context. With this information, we may know more challenges of applying data compression to IoT data in practice. Second, where is the best point to deploy a data compression solution in the I/O stack. This feedback will help us deliver a really practical solution.

Controversial points: Edge-to-cloud data transfer usually happens when the edge does not have enough computation resources for in-depth data analysis, and data has to be moved to the cloud for analysis. We propose data compression to accelerate edge-to-cloud data transfer, but data compression is compute-intensive. How to coordinate system resources for data compression and other edge tasks can be a challenge.

The type of discussion this paper may generate: This paper will likely generate discussion regarding the benefits and pitfalls of data compression on edge servers. The tens of IoT-type datasets we screen out of hundreds of public datasets may attract some attendees. Our adaptive compressor selection in resource limitation environments can also inspire discussion, considering the wide use of data compression techniques in a lot of storage application scenarios. Considering the popularity of general compressors such as *gzip* and *zstd*, there must be some researchers who have conducted similar research, we expect both challenges and consonance from these peers.

Remaining open issues: We have demonstrated that compressors for a specific data type usually can achieve a much better compression performance than general compressors. However, some specific compressors are actually based on lossy compression. For example, *SZ* can well compress floating-point data, but there is accuracy loss of the data. How to ensure user tolerable accuracy loss is an open issue. Another issue is that our current adaptive compression scheme is a greedy algorithm, instead of a theoretically optimal solution. Whether there is a theoretically optimal solution for our problem domain is another open issue.

Under what circumstances the whole idea might fall apart: If there exists a compressor that can beat all other peers and perform the best on most datasets in most conditions, then adaptive compression will not bring much benefit. If in an environment where network bandwidth is higher than the compression throughput, then adaptive compression will not bring much benefit.

Acknowledgments

We are grateful to our shepherd Huiping Cao and the anonymous reviewers for their insightful feedback. Wen's work was partly supported by National Natural Science Foundation of China under Grant No. 61972441; the Shenzhen Science and Technology Program under Grant No. JCYJ20190806143405318.

References

[1] Aviation weather center real-time data. https://aviationweather.gov/dataserver. Accessed: 2019-12-13.

[2] Awesome public datasets. https://github.com/awesomedata/awesome-public-datasets. Accessed: 2019-11-15.

[3] data-viz-challenge. https://github.com/localytics/data-viz-challenge. Accessed: 2020-1-5.

[4] Imos - australian national mooring network (anmn) - ctd profiles. https://catalogue-imos.aodn.org.au. Accessed: 2019-12-27.

[5] Iot signals report: Iot's promise will be unlocked by addressing skills shortage, complexity and security. https://blogs.microsoft.com/blog/2019/07/30. Accessed: 2020-01-15.

[6] Nssdca photo gallery. https://nssdc.gsfc.nasa.gov/photo_gallery. Accessed: 2020-1-17.

[7] Tool to convert dicom to dicom. http://gdcm.sourceforge.net/wiki/index.php/Gdcmconv. Accessed: 2019-12-16.

[8] Webp compression techniques. https://developers.google.com/speed/webp/docs/compression. Accessed: 2019-11-21.

[9] Ragaad Altarawneh, Christina Strong, Luis Remis, Pablo Muñoz, Addicam Sanjay, and Srikanth Kambhatla. Navigating the visual fog: Analyzing and managing visual data from edge to cloud. In *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[10] George Amvrosiadis, Ali R Butt, Vasily Tarasov, Erez Zadok, Ming Zhao, Irfan Ahmad, Remzi H Arpaci-Dusseau, Feng Chen, Yiran Chen, Yong Chen, et al. Data storage research vision 2025: Report on nsf visioning workshop held may 30–june 1, 2018. 2018.

[11] Joseph Azar, Abdallah Makhoul, Mahmoud Barhamgi, and Raphaël Couturier. An energy efficient iot data compression approach for edge machine learning. *Future Generation Computer Systems*, 96:168–175, 2019.

[12] Madhumita Bharde, Suparna Bhattacharya, Dileep Deepa Shree, et al. Store-edge ripplestream: Versatile infrastructure for iot data transfer. In *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[13] Davis Blalock, Samuel Madden, and John Guttag. Sprintz: Time series compression for the internet of things. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(3):1–23, 2018.

[14] Youdong Chen, Qiangguo Feng, and Weisong Shi. An industrial robot system based on edge computing: An early experience. In {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[15] Kenneth Clark, Bruce Vendt, Kirk Smith, John Freymann, Justin Kirby, Paul Koppel, Stephen Moore, Stanley Phillips, David Maffitt, Michael Pringle, et al. The cancer imaging archive (tcia): maintaining and operating a public information repository. *Journal of digital imaging*, 26(6):1045–1057, 2013.

[16] Sheng Di and Franck Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 ieee international parallel and distributed processing symposium (ipdps)*, pages 730–739. IEEE, 2016.

[17] Ali E Elgazar, Mohammad Aazam, and Khaled A Harras. {SMC}: Smart media compression for edge storage offloading. In *2nd* {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[18] Harshit Gupta, Zhuangdi Xu, and Umakishore Ramachandran. Datafog: Towards a holistic data management platform for the iot age at the network edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[19] Danny Harnik, Ronen Kat, Dmitry Sotnikov, Avishay Traeger, and Oded Margalit. To zip or not to zip: Effective resource usage for real-time compression. In *Presented as part of the 11th* {*USENIX*} *Conference on File and Storage Technologies (*{*FAST*} *13)*, pages 229–241, 2013.

[20] Yingting Jin, Yuzhuo Fu, Ting Liu, and Lan Dong. Adaptive compression algorithm selection using lstm network in column-oriented database. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 652–656. IEEE, 2019.

[21] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Rethinking adaptability in wide-area stream processing systems. In *10th* {*USENIX*} *Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, 2018.

[22] Antti Lehmussola, Pekka Ruusuvuori, Jyrki Selinummi, Heikki Huttunen, and Olli Yli-Harja. Computational framework for simulating fluorescence microscope images with cell populations. *IEEE transactions on medical imaging*, 26(7):1010–1016, 2007.

[23] Tao Lu, Qing Liu, Xubin He, Huizhang Luo, Eric Suchyta, Jong Choi, Norbert Podhorszki, Scott Klasky, Mathew Wolf, Tong Liu, et al. Understanding and modeling lossy compression schemes on hpc scientific data. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 348–357. IEEE, 2018.

[24] Ioannis Psaras, Onur Ascigil, Sergi Rene, George Pavlou, Alex Afanasyev, and Lixia Zhang. Mobile data repositories at the edge. In {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[25] Andreas Reinhardt, Paul Baumann, Daniel Burgstahler, Matthias Hollick, Hristo Chonov, Marc Werner, and Ralf Steinmetz. On the accuracy of appliance identification based on distributed load metering data. In *2012 Sustainable Internet and ICT for Sustainability (SustainIT)*, pages 1–9. IEEE, 2012.

[26] Felix M. Riese and Sina Keller. Introducing a Framework of Self-Organizing Maps for Regression of Soil Moisture with Hyperspectral Data. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 6151–6154, Valencia, Spain, July 2018.

[27] Biljana Risteska Stojkoska and Zoran Nikolovski. Data compression for energy efficient iot solutions. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–4. IEEE, 2017.

[28] Ricardo Emanuel Vaz Vargas, Celso José Munaro, Patrick Marques Ciarelli, André Gonçalves Medeiros, Bruno Guberfain do Amaral, Daniel Centurion Barrionuevo, Jean Carlos Dias de Araújo, Jorge Lins Ribeiro, and Lucas Pierezan Magalhães. A realistic and public dataset with rare undesirable real events in oil wells. *Journal of Petroleum Science and Engineering*, 181:106223, 2019.

[29] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[30] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *7th* {*USENIX*} *Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.