Applied Laser Technology
Vol. 31, No.2 June (2024), pp.01–12

VERGE DB: AN IOT ANALYTICS DATABASE FOR EDGE DEVICES

Mr.Pitta Sankara Rao[1]., Jagadhabi Tanuja[2]
1 Assistant Professor, Department of ECE, Malla Reddy College of Engineering for Women.,

Maisammaguda., Medchal., TS, India

2, B.Tech ECE (21RG1A0427),

Malla Reddy College of Engineering for Women., Maisammaguda., Medchal., TS, India

ABSTRACT

New infrastructure is needed to gather, store, and analyze massive amounts of time-series data generated by the explosion of Internet of Things (IoT) applications. We anticipate that considerable data processing will need to occur at the edge of the network in order to achieve these scalability requirements. In this work, we introduce VergeDB, a database designed for adaptive and task-aware compression of Internet of Things data, which treats complicated analytical tasks and machine learning as first-class operations. Depending on the context, VergeDB may act as either a lightweight storage engine that compresses the data depending on downstream duties or as an edge-based database that handles compression as well as in-situ analytics on raw and compressed data. VergeDB will make choices to improve speed, data compression, and downstream job correctness by making the most of the available CPU resources, storage space, and network bandwidth.

INTRODUCTION

New monitoring applications made possible by the expanding Internet of Things (IoT) are reshaping whole sectors including the automotive, agricultural, healthcare, retail, manufacturing, transportation, and utility sectors [38]. Analysts predict that by 2020 there will be billions of IoT devices in use, each producing zettabytes (ZB) of data [17, 22]. New difficulties arise in data collecting, processing, storage, and analysis as a result of this influx of data, which consists mostly of time series [3, 53].

Although analytics over sensor networks was widely investigated by the database community in the early 2000s [51], the hardware and software assumptions behind this study have since undergone significant changes. To begin, edge devices have greatly expanded their capabilities over the last two decades, allowing for considerably more complicated processes to be performed on the data before it is centralized. Furthermore, there has been a transition in analytics workloads away from SQL-based analytics and toward machine learning, particularly for time-series data. We propose that the data being collected by the Internet of Things management systems do not adequately embrace these two transitions in computing.

Applied Laser Technology
Vol. 31, No.2 June (2024), pp.01–12

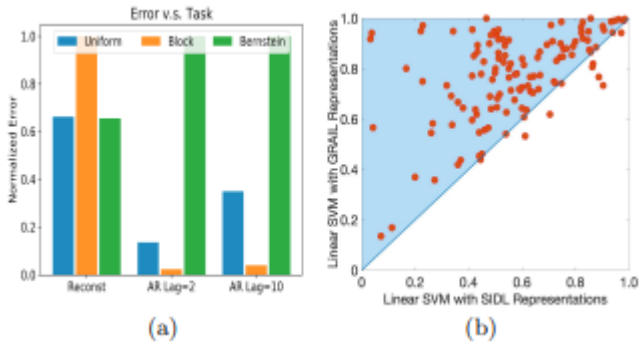


Figure 1: Effectiveness of data reduction methods on two downstream tasks: (a) autoregressive modeling of a timeseries of particles using different subsampling methods; and (b) classification accuracy of 128 time-series datasets [11] using similarity-preserving representation learning methods.

This is a difficult puzzle to solve since the two parts are deeply intertwined. Edge devices with sufficient processing power can make nuanced judgments about I private data that should be destroyed, (ii) data relevant for triggering actions that should be evaluated in real time, and (iii) historical or uninteresting data that should be aggressively compressed or aggregated. Machine learning data consumers, on the other hand, are very sensitive to artifacts in the data collecting process that alter the distributions of the features that are collected. Thus, little modifications to data compression or aggregation may have significant effects on task precision [48]. Choices must be made with the downstream job in mind, since moving processes to edge devices may minimize the need for compute, storage, memory, power, and bandwidth in the central processing unit (CPU). An edge-based system that can swiftly ingest data from sensors while optimizing for compression, aggregation, and filtering depending on the demands of a downstream analytics consumer is something that is sorely lacking in the realm of IoT data management.

Accordingly, this paper presents VergeDB, a database for adaptive and task-aware compression of IoT data that supports complex analytical tasks and machine learning as firstclass operations. VergeDB allows for either lightweight storage engines that compress the data based on downstream tasks or for edge-based databases that manage both compression and in-situ analytics on raw and compressed data. One should think of VergeDB as an intermediary between the “fire hose” of IoT data and user-written analytics programs. VergeDB will optimize for available computation resources, storage capacity, and bandwidth when making decisions in order to maximize throughput, data compression, and result quality, while adhering to resource constraints.

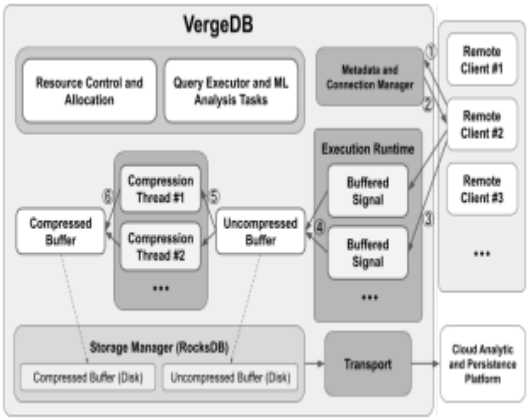


Figure 2: System overview.

While the database community has developed an extensive theory on how to appropriately select data compression schemes [19], much less attention has been given in their effectiveness for typical IoT analytics tasks, e.g., anomaly detection. As a result, existing IoT solutions suffer from three main drawbacks: (i) solutions rely on lossless compression methods that do not support directly any form of analytics (e.g., byte-oriented compression methods that require full decompression before any operation can be performed [55]) or support lightweight columnar encoding that has limited benefits for numeric data types; (ii) solutions rely on lossy compression methods tailored to support only specific operations (e.g., sampling methods supporting aggregation operations [1, 9]); or (iii) solutions rely on

quantization and indexing mechanisms that cannot easily be extending to machine learning tasks (e.g., popular time-series databases[1]).

To illustrate this point, in Figure 1 we present two examples of the effectiveness of compression methods in a autoregressive modelling task and a classification task. First, we show how new non-uniform sampling methods [27] and uniform sampling may be more accurate than a block sampler (Section 4.2) in terms of reconstruction error, but can be significantly worse when compared in terms of end-to-end performance on an autoregressive model (Figure 1a). Second, we demonstrate how a recent time-series sparse coding method, SIDL [54], that is known it can support aggregation queries accurately [35], fails to accurately support timeseries classification unlike GRAIL [40] across 128 time-series datasets of the UCR time-series archive [11] (each circle represents a dataset in Figure 1b and circles above diagonal show better classification accuracy for GRAIL).

VergeDB is an important step to move beyond the basic SQL-like analytics towards anomaly detection, regression, clustering, and classification. VergeDB sits between the source of data and a downstream analytics consumer that optimizes which data to collect, how accurately to represent it, and how to allocate edge resources. By pushing algorithms to the edge devices, VergeDB alleviates the strain of centralized IoT solutions that can have deteriorating performance with the increasingly larger number of IoT devices and data. However, as our motivating results suggest, the task must inform these decisions.

SYSTEM OVERVIEW

In Figure 2, we outline the key components of VergeDB. Our initial prototype is written in Rust. We elaborate on the compression methods in more detail in Section 4.

VergeDB allows the collection and aggregation of data from multiple device clients. The database can be configured to accept multiple different signals that corresponds to a metric that the clients produce, such as temperature and humidity (1 and 2). For each signal, a server-buffered signal is created when initialized. The server accepts data generated by the remote clients (3), and the buffered signal segments the data into fixed size arrays, attaches a timestamp for the segment, and pushes it to the uncompressed buffer (4). As the uncompressed buffer is being filled, compression threads offload data from the uncompressed buffer and process them (5). The compression threads are adaptively configured to use different compression algorithms based on storage capacity, network bandwidth, ingestion rate, and specified analytical task. If the uncompressed buffer exceeds its capacity, which may happen when the ingestion rate exceeds the compression speed, the data is flushed to the disk. The compression threads push the compressed data into a compressed buffer pool, which can also flush to disk (6). VergeDB can execute queries or analysis (e.g., clustering or outlier detection) either over the compressed data or the raw time-series segments in the uncompressed buffer.

VergeDB currently supports byte-compression techniques, such as Gzip and Snappy, and lightweight encodings, such as dictionary encoding, Gorilla [44], and Sprintz [5] for numeric data. VergeDB also supports specialized time-series representation methods (i.e., lossy compression), such as Piecewise Aggregate Approximation (PAA) [52, 28], Fourier transform [15], and sparse and dense representation learning methods [54, 40] for advanced analytical workloads. These approaches differ in terms of compression ratio, compression throughput, and query efficiency and accuracy. There is no one size fits all approach for any time series or task. With system resources limitations found in edge devices, such a system must be able to adaptively switch the compression approach according to the data features (e.g., data arriving rate, sortedness, and cardinality) and target tasks. For example, by using principled properties of the Fourier transform, we can effectively control computation and memory usage during training and inference for Convolutional Neural Networks, while retaining high prediction accuracy and improving robustness to adversarial attacks [13]. Compression should not only reduce storage requirements, but also enhance the query performance. Most current numeric compression methods are not query-friendly as data need to be uncompressed before query execution. Such decompression introduces unnecessary overhead for a query which counters the goal of an efficient edge database. Therefore, more powerful and

efficient compression methods for time-series data are needed to not only compress the data effectively but also run queries directly on encoded data under limited edge resources. In Section 4.1, we present a precision-aware compression method that works on numeric data of bounded range, which accelerates filtering operations while achieving competitive compression ratio and throughput. In Section 4.3, we present a solution to quantize learned similarity-preserving representations in order to enable a multitude of data mining and machine learning tasks to efficiently retrieve similar time series even under resourceconstrained settings. Learning representations that preserve arbitrary, user-defined, similarity functions is crucial considering recent studies that have demonstrated significant trade-offs between accuracy and efficiency for downstream tasks based on the choice of similarity function [40, 43].

In addition to multiple compression techniques, the system also allows for data subsampling methods. Like compression, there is no single sampling scheme that is universally optimal for a given dataset and analytics. We find that uniform sampling is effective at capture global properties (e.g., overall mean), but ineffective at capturing local trends (e.g., those that would be useful features in an anomaly detector) so we develop new alternatives (Section 4.2). Subsamples could be further compressed by the aforementioned methods in a hierarchical fashion.

Beyond numeric data, VergeDB also supports string attributes that are often associated (as metadata) with timevarying measurements. Recent projects on string compression show impressive compression and query benefits, especially for attributes sharing a distinctive pattern (e.g., IP address, log tag, and location coordinates) [26, 7, 34].

Currently, VergeDB permits each signal to be associated with one or more compression schemes so that multiple applications can be supported. We are working on a controller to automatically select the compression approach, given the workload, data arrival rates, and resource capacity.

RELATED WORK

We refer the reader to [14] for an extensive overview of representation methods for time series and to [25] for a survey on time-series database management systems.

Approximate query processing: Approximate query processing (AQP) is a widely studied paradigm for accelerating computation by enabling analytics over some form of compressed data. Based on the error guarantees, we divide AQP methods into probabilistic and deterministic methods. Probabilistic methods approximate the query answers over a small sample of the data and provide some confidence value for the approximated answer [1, 9, 39, 50]. In contrast, deterministic methods offer approximate answers with perfect confidence. The majority of the work has focused on supporting aggregation operations for a single time series [10, 20] with the exception of recent work that focuses on correlation-based queries for multiple time series [32].

Data compression: Database systems rely on data compression techniques (e.g., histograms) [23, 45] to estimate the cardinality [23] and selectivity [46] of specific queries. Unfortunately, such summarization techniques are not suitable for time-series data. The signal processing and time-series communities has devoted significant effort to study representation methods that reduce the high dimensionality of time series and lower the storage and computational costs.

Time-series representations: The most prevalent techniques in that context represented time series using Singular Value Decomposition (SVD) [30, 47], Discrete Fourier Transform (DFT) [2, 15], and Discrete Wavelet Transform (DWT) [8]. The Piecewise Aggregate Approximation (PAA) [52, 28] represents time series as mean values of segments, whereas other approaches, namely, Piecewise Linear Approximation (PLA) [49] and Adaptive Piecewise Constant Approximation (APCA) [29], fit a polynomial model or use a constant approximation for each segment, respectively. The output of all previous methods is numeric. Symbolic methods additionally quantize the numeric output. For example, the Symbolic Aggregate approXimation (SAX) [33] and rely on alphabets to transform PAA epresentations into short words. A number of works exist that rely on dictionary-based compression methods to support more advanced analytics, such as classification and

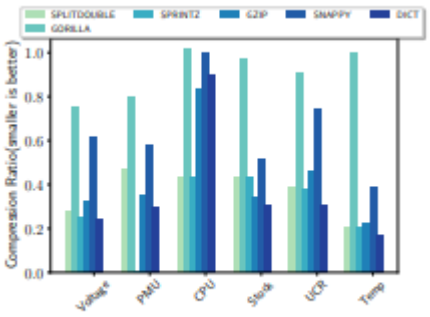
similarity search. For example, in [36], a data-aware version of PAA uses vector quantization to constructa codebook of segments and develop a multi-resolution symbolic representation to support similarity search queries. In [31], a Limpel-Zivdictionary-based compression method for time series is proposed, which can be used for time-series classification [21]. Tristan [35], approximates time series as a combination of time-series patterns using an extracted dictionary. The sparse weights (or coefficients) that correspond to each atom in the dictionary serve as the new compressed representation of the time series.

In addition to high-level time-series representations, there are many compression techniques for numeric data, which perform differently based on the time-series distribution.

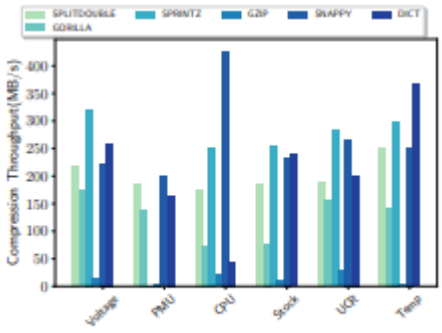
Record-oriented compressiontransform each record into a compact representation. Popular record-oriented compression includes bit-packed encoding, delta encoding, runlength encoding, dictionary encoding and their hybrid variations for integer types, and Gorilla [44] and Sprintz [5] for float types. Bit-packed encoding stores input value using as few bits as possible to save space. Delta encoding saves delta between consecutive records. Run-length encoding saves consecutive repeated records with tuple. Dictionary encoding [34] uses bijective mapping to replace records with a more compact code. Gorilla [44] is an in-memory time-series database developed by Facebook. It introduces two encoding to improve delta encoding: deltaof-delta for timestamps, which is usually a increasing integer sequence. XOR-based encoding for value domain, which is float type. In the XOR-based float encoding, successive float values are XORed together, and only the different bits (delta) are saved. The delta is then stored using control bits to indicate how many leading and trailing zeroes are in the XOR value. Gorilla is the state-of-the-art approach for float data compression. Sprintz [5] was originally designed for integer time-series compression. Sprintz employs a forecast model to predict each value based on previous records. Sprintz then encodes the delta between the predicted value and the actual value. Those delta values are usually closer to zero than the actual value, making it smaller when encoded with bit-packed encoding. It is also possible to apply Sprintz to floats by first quantizing the float into integer. Those record-oriented compression maintains entry boundaries during compression, which enables direct access and filtering on compressed records without decoding.

Byte-oriented compression encodes the data stream in byte level. Popular techniques, such as Gzip and Snappy, derived from the LZ77 family [55], which looks for repetitive sub-sequence within a sliding window on the input byte stream, and encodes the recurring sub-sequence as a reference to its previous occurrence. For better compression ratio, Gzip applies Huffman encoding on the reference data stream. Snappy skips Huffman encoding for higher throughput. Byte-oriented compression treats the input values as byte stream and encodes them sequentially. The data block needs to be decompressed before accessing original values.

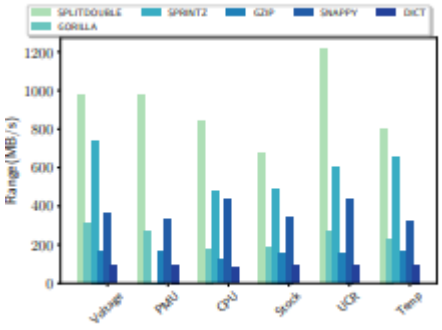
Time-series subsampling: Time-series subsampling has been extensively investigated from a theoretical perspective[12, 37], with several efficient algorithms and known guarantees. However, most practical systems today choose a uniform sampling scheme where data points are sampled at a known interval, and there is substantial work on how to adaptively tune that interval [18]. Our experiments suggest that uniform sampling, even when adaptive, is not a panacea and can loose local structure. Non-uniform sampling work includes BlazeIt, a database system that uses Empirical Bernstein [27] to subsample time series data, we find that such a method is effective at counting events but not as good for trend estimation. In VergeDB, we develop new adaptive sampling schemes that are better tuned for machine learning data consumers.



(a) Compression Ratio



(b) Compression Throughput



(c) Range Filtering

Figure 3: Compression performance on representative datasets: Voltage generated by electro-mechanical energy conversion. PMU recording synchrophasor events in a power grid.CPU usage data from the Azure public dataset.Stock including daily price for all US-based stocks.UCR time series classification archive. Temp including daily temperature of major cities.

SYSTEM COMPONENTS We outline three key novel compression methods for supporting analytics in VergeDB. Individually, these methods support a wide range of database operations, but the methods are complementary and can be composed to support a wider range of advanced analytical operations and machine learning tasks. For the similarity-preserving compression method, GRAIL, a preliminary report is available [40].

Precision-Aware Data Compression The main goal of compression for time-series databases is to build an efficient and effective compression approach for numeric data. The compression should not only keep the compressed data size smaller, but also support fast query execution, which includes fast filtering, aggregations, distancebased similarity evaluations, and materialization for machine learning tasks. We start by studying the aforementioned compression techniques on a wide range of datasets that cover common IoT use cases.

We measure compression ratio, compression performance, and range filtering on float values only, as shown in Figure 3. The state-of-the-art approach, Gorilla, does not achieve satisfactory results on most datasets. On the CPU dataset, the compressed data is even larger than the original one. Gorilla performs poorly on those datasets as floats provides more than required precision which leads to significant bit changes even with a subtle value change. In terms of compression throughput, as shown in Figure 3b, Sprintz achieves a good compression throughput overall, but it fails on thePMU dataset where float quantizing introduces integer overflow issue. While Snappy has high compression throughput and range filtering for many datasets, it does not compress effectively on most datasets. As expected, GZip exhibits the inverse behavior. Dictionary encoding (Dict) works well the cardinality is low enough that recoding float values as integer keys can provide good compression. Query performance (i.e., range filtering) varies with different compression techniques as shown in Figure 3c. Sprintz achieves high filtering throughput overall, since we can partially avoid decoding the encoded value by predicate rewriting. The float predicate value is translated (quantized) into the target integer before the filter execution, but Sprintz still needs to decode the value into an integer for filter evaluation. Except for Sprintz, full decompression is needed for all the other approaches when filtering values.

A good numeric data compression should be able to work directly on encoded data directly without decoding to speed up query performance. However, the amount of precision provided with standard float or double formats limits either the compression ratio or throughput performance. Given that many IoT domains work on devices with bounded precision (such as a thermometer giving 1 or 2 decimal points of precision), we are developing a new compression technique, SplitDouble, to work on bounded range data with fixed precision (such as numeric in PostgeSQL) for float types. We decompose the data into integers and decimals with a columnar format, and use a combination of bit-packing and delta encoding. As shown in Figure 3, SplitDouble provides good compression ratios, compression throughput, and

range filtering. We are currently exploring methods to improve the compression throughput and query support for this method.

Subsampling Compression

Subsampling can be a powerful tool to reduce the amount of data to be stored and analyzed. However, as we saw in the introduction, existing methods can fail to capture temporal structure. We bring ideas from time-series bootstrap [12] to design a new time-series subsampler that captures local trend information more accurately than popular alternatives by sampling contiguous blocks of data. Ideally, this method should be able to subsample blocks from the data and stop when the a convergence criteria has been met. There aretwo important pieces of the method that are vital to its effectiveness: the block size and the stopping criteria. We can build an entire family of samplers by using different criteria for stopping conditions and block size parameters (we defer this discussion to future work).

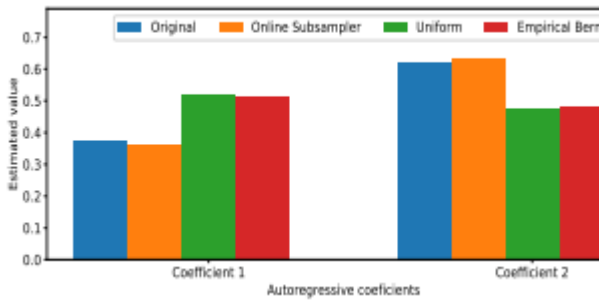


Figure 4: Synthetic generated AR(2) time series and coefficients retrieved from 0.35% of the data using different subsampling methods

We applied the block subsampling method to a sythetic time series generated from a autoregressive model of order

2. The goal in this particular scenario is to estimate two autoregressive coefficients of the stochastic process. Obtaining good estimates of coefficients in an autogressive process is extremely useful in tasks ranging from prediction of future events to simulating the original stochastic process. Based on Figure 4, the results of the block subsampling are far superior to the other methods. They are within 5% of the ground truth and in this particular case using only 0.35% of the original data. All subsampling methods were controlled by the number of data points that they would generate, and the accuracy can be indirectly controlled by the convergence rate of the subsampling method.

Similarity-Preserving Compression

The most accurate time-series mining methods cannot often scale to millions of time series [14, 4]. This is because, in addition to large time-series volumes, the temporal ordering and high-dimensionality complicates the comparison of time series and increases the computation and storage requirements of methods operating directly over time series. The design of effective solutions require decisions for three core components [14]: (i) the representation method to compress time series; (ii) the comparison method to determine the similarity between time series and (iii) the indexing method to organize and retrieve similar time series from massive collections. Unfortunately, these components have largely been investigated and developed independently [40], often resulting in mutually incompatible methods.

To address this issue, we recently presented the GRAIL framework [40] to automate the process of learning how to compress time series while preserving user-specified similarity functions. This differs substantially from current approaches (see Section 3) that are agnostic to the similarity function needed in downstream tasks. In Figure 1b, we illustrated this point by presenting a classification experiment. Specifically, when a common classifier operates over GRAIL's representations achieves substantially better classification accuracy in comparison to when it operates over compressed data generated with a task-agnostic method.

With GRAIL, we coupled two out of the three core components mentioned before, namely, the representation method with the comparison method. An important next step is to couple a method that quantizes numeric vectors and accelerates similarity search. Such a method, in combination with GRAIL's representations that preserves time-series similarities will enable data mining and machine learning tasks (e.g., clustering, classification, pattern search, anomaly detection, sampling, and visualization [41, 42, 40]) to efficiently retrieve similar time series even under

resource-constrained settings. We are actively exploring such quantization methods. In particular, we focus on methods that rely on clustering to partition the search space. In simple terms, every vector is associated with a handful of representative examples and, therefore, when a new time series arrives, it suffices to compare only against such representative examples, which avoids looking up the vast portion of dissimilar time series. In Figure 5 we report preliminary results of our Clustered GRAIL (C-GRAIL) method against rival approaches for numeric vector quantization. We measure the precision accuracy of retrieving the true nearest neighbor as well as the runtime cost it takes to achieve that. We observe that C-GRAIL achieves comparable runtime performance to one of the fastest but less accurate methods, namely, Bolt [6]. Interestingly, our method outperforms in terms of accuracy two state-of-the-art product quantization methods, namely, PQ [24] and OPQ [16].

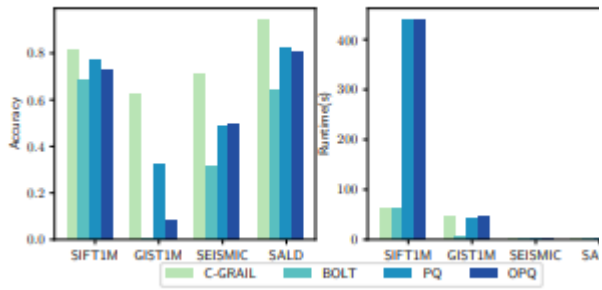


Figure 5: Precision (accuracy) and runtime performance of quantization methods that accelerate similarity search on established benchmark datasets [6]

PROTOTYPE SYSTEM EVALUATION

To evaluate the ingestion throughput of our prototype implementation, we set up an experiment where a remote client (residing in the same datacenter) is sending messages over the network to VergeDB. The batch size of each message was fixed to 64 points of 64-bit float values and, overall, 400M points were received. We measure the time required to complete the ingestion, from which the throughput was calculated. We repeat the experiment 10 times and report the results without compression and with two popular compression methods, namely, GZip and PAA, in Figure 6. We observe an increase in the throughput as we vary the number of threads in all three configurations. The sublinear growth is due to the increasing lock contention. Interestingly, for PAA, the compression thread completes each segment faster than in the case of GZip, causing a greater contention on the uncompressed bufferpool lock. By having a closer look at the compression rates, PAA compresses almost 100% of the segments while maintaining on ingestion rate of over 1.8M points/sec even with only one thread. In contrast,GZip compresses on average 1% of the ingested segments. VergeDB is able to ingest data for multiple signals without loss in the overall throughput. By adaptively changing the number of available threads for compression based on the input data rates and compression cost, VergeDB could eventually control the rate at which data is being compressed in order to minimize storing uncompressed data albeit at the cost of some ingestion throughput.

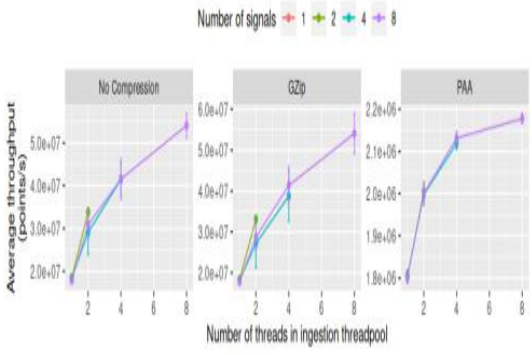


Figure 6: Average ingestion throughput of VergeDB without compression and with two popular compression methods.

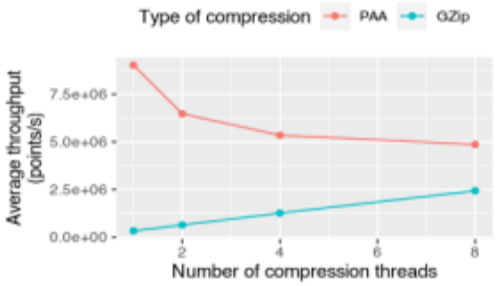


Figure 7: Compression throughput for two popular compression methods with up to 8 compression threads

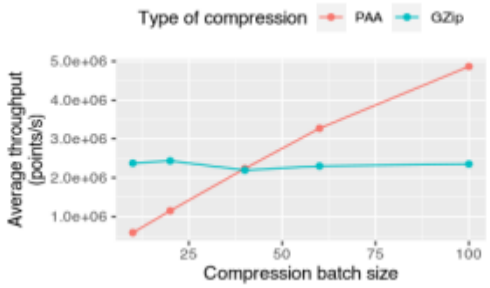


Figure 8: Compression throughput for two popular compression methods with different compression batch sizes

Figures 7 and 8 show the compression throughput while varying the number of compression threads and compression batch sizes (number of segments fetched per lock), respectively. For GZip, we observe that by increasing the number of compression threads, the compression throughput improves while increasing the compression batch size does not help because GZip is a computation-intensive compression workload. On the other hand, PAA is much faster than GZip so that lock contention becomes the bottleneck. By increasing the compression batch size, PAA achieves significant improvement in compression throughput while adding more compression threads does not result in an improvement but adds locking overhead. These preliminary results demonstrate VergeDB's ability to ingest high throughput data and the importance of compression selection while adhering to available resources.

CONCLUSION

The volume of information created by connected devices and requiring cloud storage and analysis is expected to increase in tandem with the popularity of the Internet of Things. To reduce data transmission to the cloud while maintaining local support, an edge-based database equipped with adaptive compression and the ability to perform advanced analytics is necessary. When implementing lossy compression methods, such adaptation will have to take into account both the available resources and the jobs that lie in the pipeline. As a beginning step towards this goal, we introduced VergeDB.

REFERENCES

[1] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In Proceedings of the 8th ACM European Conference on Computer Systems, pages 29–42. ACM, 2013.

[2] R. Agrawal, C. Faloutsos, and A. Swami.Efficient similarity search in sequence databases. FODA, pages 69–84, 1993.

[3] G. Amvrosiadis, A. R. Butt, V. Tarasov, E. Zadok, M. Zhao, I. Ahmad, R. H. Arpaci-Dusseau, F. Chen, Y. Chen, Y. Chen, et al. Data storage research vision 2025: Report on nsf visioning workshop held may 30–june 1, 2018. 2018.

[4] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery, 31(3):606–660, 2017.

[5] D. Blalock, S. Madden, and J. 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.

[6] D. W. Blalock and J. V. Guttag. Bolt: Accelerated data mining with fast vector compression. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 727–735, 2017.

[7] P. Boncz, T. Neumann, and V. Leis.Fsst: fast random access string compression. Proceedings of the VLDB Endowment, 13(12):2649–2661, 2020.

[8] K.-P. Chan and A. W.-C.Fu.Efficient time series matching by wavelets. In ICDE, pages 126–133. IEEE, 1999.

[9] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. ACM Transactions on Database Systems (TODS), 32(2):9, 2007.

[10] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In 21st International Conference on Data Engineering (ICDE'05), pages 20–31.IEEE, 2005.

[11] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML. The ucr time series classification archive, October 2018. https: //www.cs.ucr.edu/˜eamonn/time series data 2018/.

[12] M. W. Dimitris N. Politis, Joseph P. Romano. Subsampling.Springer Series in Statistics.Springer, 1 edition, 1999.

[13] A. Dziedzic, J. Paparrizos, S. Krishnan, A. Elmore, and M. Franklin.Band-limited training and inference for convolutional neural networks.In International Conference on Machine Learning, pages 1745–1754, 2019.

[14] P. Esling and C. Agon.Time-series data mining. ACM Computing Surveys (CSUR), 45(1):12, 2012.

[15] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos.Fast subsequence matching in time-series databases. In SIGMOD, pages 419–429, 1994.

[16] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. IEEE transactions on pattern analysis and machine intelligence, 36(4):744–755, 2013.

[17] S. George. IoT Signals report: IoT's promise will be unlocked by addressing skills shortage, complexity and security., 2019 (accessed August 15, 2020). https://blogs.microsoft.com/blog/2019/07/30/.

[18] D. Giouroukis, A. Dadiani, J. Traub, S. Zeuch, and V. Markl. A survey of adaptive sampling and filtering algorithms for the internet of things. In DEBS'20: 14th ACM International Conference on Distributed and Event-Based Systems, 2020.

[19] G. Graefe and L. D. Shapiro.Data compression and database performance.University of Colorado, Boulder, Department of Computer Science, 1990.

[20] M. Greenwald, S. Khanna, et al. Space-efficient online computation of quantile summaries. ACM SIGMOD Record, 30(2):58–66, 2001.

[21] B. Hu, Y. Chen, and E. Keogh.Time series classification under more realistic assumptions. In Proceedings of the 2013 SIAM International Conference on Data Mining, pages 578–586. SIAM, 2013.

[22] M. Hung. Leading the iot, gartner insights on how to lead in a connected world. Gartner Research, pages 1–29, 2017.

[23] Y. E. Ioannidis and V. Poosala.Balancing histogram optimality and practicality for query result size estimation. In AcmSigmod Record, volume 24, pages 233–244. ACM, 1995.

[24] H. Jegou, M. Douze, and C. Schmid.Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence, 33(1):117–128, 2010.

[25] S. K. Jensen, T. B. Pedersen, and C. Thomsen. Time series management systems: A survey. IEEE Transactions on Knowledge and Data Engineering, 29(11):2581–2600, 2017.

[26] H. Jiang, C. Liu, Q. Jin, J. Paparrizos, and A. J. Elmore.Pids: attribute decomposition for improved compression and query performance in columnar storage. Proceedings of the VLDB Endowment, 13(6):925–938, 2020.

[27] D. Kang, P. Bailis, and M. Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics, 2018.

[28] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. Knowledge and information Systems, 3(3):263–286, 2001.

[29] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra.Locally adaptive dimensionality reduction for indexing large time series databases. ACM Sigmod Record, 30(2):151–162, 2001.

[30] F. Korn, H. V. Jagadish, and C. Faloutsos.Efficiently supporting ad hoc queries in large datasets of time sequences. In SIGMOD, SIGMOD '97, pages 289–300, New York, NY, USA, 1997. ACM.

[31] W. Lang, M. Morse, and J. M. Patel.Dictionary-based compression for long time-series similarity. IEEE transactions on knowledge and data engineering, 22(11):1609–1622, 2009.

[32] C. Lin, E. Boursier, and Y. Papakonstantinou. Plato: approximate analytics over compressed time series with tight deterministic error guarantees. Proceedings of the VLDB Endowment, 13(7):1105–1118, 2020.

[33] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, pages 2–11. ACM, 2003.

[34] C. Liu, M. Umbenhower, H. Jiang, P. Subramaniam, J. Ma, and A. J. Elmore.Mostly order preserving dictionaries. In 2019 IEEE 35th InternationalConference on Data Engineering (ICDE), pages 1214–1225. IEEE, 2019.

[35] A. Marascu, P. Pompey, E. Bouillet, M. Wurst, O. Verscheure, M. Grund, and P. Cudre-Mauroux. Tristan: Real-time analytics on massive time series using sparse dictionary compression. In 2014 IEEE International Conference on Big Data (Big Data), pages 291–300. IEEE, 2014.

[36] V. Megalooikonomou, Q. Wang, G. Li, and C. Faloutsos.A multiresolution symbolic representation of time series. In ICDE, pages 668–679. IEEE, 2005.

[37] V. Mnih, C. Szepesv´ari, and J.-Y.Audibert. Empirical bernstein stopping. In Proceedings of the 25th International Conference on Machine Learning, ICML ˆaA˘Z08, page 672ˆa ´ A¸S679, New York, NY, USA, 2008. ˘ Association for Computing Machinery.

[38] I. C. Ng and S. Y. Wakenshaw. The internet-of-things: Review and research directions. International Journal of Research in Marketing, 34(1):3–21, 2017.

[39] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie.Online aggregation for large mapreduce jobs. Proc. VLDB Endow, 4(11):1135–1145, 2011.

[40] J. Paparrizos and M. J. Franklin. Grail: efficient time-series representation learning. Proceedings of the VLDB Endowment, 12(11):1762–1777, 2019.

[41] J. Paparrizos and L. Gravano.k-shape: Efficient and accurate clustering of time series. In SIGMOD, pages 1855–1870. ACM, 2015.

[42] J. Paparrizos and L. Gravano.Fast and accurate time-series clustering. ACM Transactions on Database Systems (TODS), 42(2):8, 2017.

[43] J. Paparrizos, C. Liu, A. J. Elmore, and M. J. Franklin. Debunking four long-standing misconceptions of time-series distance measures. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pages 1887–1905, 2020.

[44] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. Proceedings of the VLDB Endowment, 8(12):1816–1827, 2015.

[45] G. Piatetsky-Shapiro and C. Connell.Accurate estimation of the number of tuples satisfying a condition. ACM Sigmod Record, 14(2):256–276, 1984.

[46] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita.Improved histograms for selectivity estimation of range predicates. In ACM Sigmod Record, volume 25, pages 294–305. ACM, 1996.

[47] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In SIGMOD, SIGMOD ’98, pages 166–176, New York, NY, USA, 1998. ACM.

[48] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems.In Advances in neural information processing systems, pages 2503–2511, 2015.

[49] H. Shatkay and S. B. Zdonik.Approximate queries and representations for large data sequences. In ICDE, pages 536–545. IEEE, 1996.

[50] L. Sidirourgos, P. Boncz, M. Kersten, et al. Sciborq: Scientific data management with bounds on runtime and quality. In CIDR, 2011.

[51] Y. Yao, J. Gehrke, et al. Query processing in sensor networks. In Cidr, pages 233–244, 2003.

[52] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. VLDB, 2000.

[53] S. Zeuch, A. Chaudhary, B. Monte, H. Gavriilidis, D. Giouroukis, P. Grulich, S. Breß, J. Traub, and V. Markl. The nebulastream platform: Data and application management for the internet of things. In Conference on Innovative Data Systems Research (CIDR), 2020.

[54] G. Zheng, Y. Yang, and J. Carbonell.Efficient shift-invariant dictionary learning.In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 2095–2104, 2016.

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