Effective Management of Time Series Data

Cristiano E. Caon
Addepar, Inc., USA
cristiano.caon@hotmail.com

Jie Li
Texas Tech University, USA
jie.li@ttu.edu

Yong Chen
Texas Tech University, USA
yong.chen@ttu.edu

2023 IEEE 16th International Conference on Cloud Computing (CLOUD) | 979-8-3503-0481-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/CLOUD60044.2023.00055

***Abstract*—Cloud computing systems, consisting of numerous nodes and components, require constant monitoring to satisfy the Quality-of-Service (QoS), making the management of large-scale time series data challenging. To address this issue, age threshold retention policies have been implemented to remove historical data, but this eliminates valuable information from older periods. In this paper, we proposed an alternative approach that applies time series deduplication with metric-based tolerance to discard readings that stabilize within a calculated tolerance window. This approach can reduce the data volume by 70.38% on average. Once the data-reduced interval is queried, the readings can be reconstructed to retrieve the original granularity with low query runtime overhead and a Mean Absolute Percentage Error of 0.74%.**

***Index Terms*—Cloud Computing, Cloud Management, System Monitoring, Time Series Data Management.**

I. Introduction

Time series data are used across a variety of scientific areas, Internet of Things (IoT) devices, climate sciences, and many others for communication and observation of changing behaviors across different intervals of time. This is especially true for cloud computing systems, which are powerful machines consisting of thousands of nodes and sub-components. By continuously monitoring various metrics such as CPU and memory usage, disk I/O, network traffic, and system uptime, system administrators can identify and resolve issues that could lead to system downtime, degraded performance, or security breaches. Additionally, monitoring can provide valuable insights into system usage patterns and trends, which can inform future system design and optimization efforts.

Dedicated databases are developed specifically for managing time series data due to their unique characteristics. These databases optimize the storage, indexing, querying, and other activities for time series, which are not optimal in other database management systems (DBMS). Considering the astonishing collection rate of time series data, time series data volume can grow extraordinarily fast. Therefore, retention policies are commonly implemented to reduce volume by deleting data based on established criteria. However, this method eliminates all valuable information that could be used for analyses across different time intervals.

In this paper, we present a time series deduplication technique using metric-based tolerance to eliminate redundancy. By analyzing time series readings collected from an data center, we observed that they stabilize at different levels for a given interval, producing close to constant readings. By creating a tolerance window around the average reading, we can ignore the redundant readings that remain constant within the tolerance window and keep only the ones that demonstrate significant changes to the system. This technique can reduce database volume by 70.38% on average. Once the reduced interval is queried, the readings can be reconstructed to retrieve the original granularity with a small query runtime overhead and a Mean Absolute Percentage Error (MAPE) of 0.74% on average. This approach allows us to maintain valuable insights while reducing the data volume size.

The contribution of this paper is mainly three-fold:

- We introduce time series deduplication as a means of reducing database volume by eliminating data redundancy.
- We present the concept of metric-based tolerance, which enhances deduplication by creating a tolerance window.
- We provide a reconstruction technique that can be applied to the reduced datasets, allowing the retrieval of the original granularity for more comprehensive analyses.
- Lastly, we propose a processing pipeline as an effective approach to managing time series data.

The rest of this paper is organized as follows. In Section II, we explain the research background and motivation. In Section III, we discuss considerations, techniques, and the processing pipeline architecture to effectively manage time series data. In Section IV, we share the experimental results. In Section V, we discuss the related works. Finally, we present the conclusions and future directions in Section VI.

II. Background & Motivation

A. Time Series Data

A time series is a series composed of data points usually sorted in increasing order of time, with time as its primary index. Each data point should contain a timestamp value (which distinguishes it from other data points in the time series) along with some numerical value that reflects the nature of the time series. The distance between two data points in the time series (i.e., the occurring frequency) can be determined by the collection rate of readings. This rate is usually configured in advance by a component responsible for data collection.

B. Time Series Databases

In contrast to conventional data found in databases, which have a one-to-one mapping to real-world objects, the signal processing community assumes that the data are merely measurements (samples) of the corresponding physical reality [1]. Due to the unique nature of time series data, dedicated databases have been developed for their management. Time

2159-6190/23/$31.00 ©2023 IEEE
DOI 10.1109/CLOUD60044.2023.00055

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

series databases address specific issues related to time series management and operations that are not optimal in relational databases, such as queries for historical data and time zone conversions. They also offer data transformations for time series and automatic calculation of basic statistics [2].

C. Age Threshold Retention Policy

Retention policies determine which data should be removed from the database according to predefined criteria, in order to eliminate unnecessary data and free up space for newer and more relevant data. There are many factors that can go into defining a retention policy criterion, but it usually directly relates to the business needs that dictate data relevance. For time series data, it is common practice to set up retention policies based on an age threshold, due to its natural relationship with time. These policies remove data that has been stored for longer than a predefined amount of time.

D. Data Source

We utilized time series monitoring data collected from an academic data center [3]. One of the partitions in the data center comprises 467 nodes, each of which includes Integrated Dell Remote Access Controller (iDRAC) components in their architecture to capture time series monitoring readings from sensors. iDRAC generates 12 metrics per node with a poll frequency of 60 seconds. We selected fan speeds, CPU and Inlet temperatures, and system power consumption for experimentation from the available metrics. The readings are then stored in the time series database TimescaleDB [4], which is hosted on a single-node machine.

E. Motivation

Simply discarding historical data in an attempt to save storage space is not a wise approach, since this data often contains valuable information that can be used to analyze the system behavior across different time periods. Many future design and configuration decisions are based on historical data, making it an important resource. Therefore, instead of applying retention policies to time series data, we seek a more sound solution that preserves the valuable data while reducing the data volume.

Our analysis of the monitoring data revealed that time series readings often remain stable and oscillate around a constant value. For example, we found that the fan speed usually remained around 9,200 rpm, with only a few cases where it reached over 10,000 rpm due to high CPU usage. From a data analytics perspective, small oscillations in such readings are not significant and can be considered the same or as duplicated values. We can take advantage of this observation by using a tolerance window to identify close readings. We can discard readings that remain relatively constant within the tolerance window and only keep the first one as representative. For consecutive readings that have differences exceeding the tolerance window, they record significant changes and should be kept. Since the monitoring data is collected at a pre-defined frequency, we can easily determine if a specific timestamp's reading has been removed and reconstruct it based on its preceding values. By carefully selecting the tolerance window and discarding loosely duplicated values, we may significantly reduce the volume of time series data.

III. Time Series Management

In this Section, we first address some design considerations. Next, we provide an architectural overview of the time series workflow and dive into the data management techniques. Lastly, we propose a processing pipeline to effectively manage the time series data.

A. Considerations

1) Deduplication: The main technique we utilized to reduce database volume is *deduplication*. In general, a typical chunk-level data deduplication system splits the input data stream into multiple data "chunks" that are each uniquely identified and duplicate-detected by a cryptographically secure hash signature (e.g., SHA-1) [5]. Data deduplication identifies the duplication and similarity of data at the chunk level and stores only one copy or part of data physically [6]. However, the definition of deduplication in our research is different than the traditional sense. Deduplication is commonly applied to entire files comparing identical chunks [7], whereas our time series deduplication focuses on the metric reading level. We provide more detail about the time series deduplication in Section III-B.

2) Cheap Storage Medium: The current trend of decreasing storage costs and the possibility of archiving the entire dataset has prompted questions about the necessity of storing subsets of the original dataset. While it may seem doable to store the entire dataset in the time series database, the performance of databases with large tables can significantly drop. This issue has been noted in previous research [8]. Even if storage costs are considered negligible, many of the cheapest solutions for storing large amounts of data are not suitable for frequent read-and-write operations of time series data, particularly when dealing with high IOPS (input/output operations per second).

3) Use Cases: We would like to address use case variations for the techniques and architecture proposed in this study since they could be tailored in many ways to best fit different scenarios.

- **Streaming vs. historical data**: We refer to the management of time series data in general, which can be more specific to either streaming or historical data. In other words, our techniques are not limited to streaming data but can also effectively reduce the data volume of historical data.
- **Data sources**: The techniques are applicable to various time series data sources beyond data centers. As an experiment, we integrated our techniques into the *MonSTer* [9], a monitoring and data management system for an HPC data center.
- **Tolerance calculation formulas**: We experimented with three formulas to calculate the metrics' tolerances, but

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

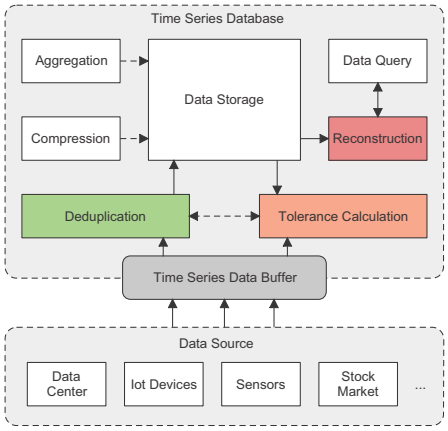

Figure 1. Architecture.

they should be tailored based on the characteristics of each time series dataset to achieve the best results.

- **Reconstruction error**: The algorithms presented in this study aim to achieve the lowest possible error, but there may be cases where more accurate reconstruction is necessary or where a higher level of error is acceptable.

B. Architecture

We start this sub-section by providing an architectural overview of the time series workflow. We then discuss our *Deduplication Simple* (DS) algorithm for time series data. Next, we talk about several formulas used for calculating the metric-based tolerance. Following that, we are able to expand the DS into the *Deduplication & Tolerance* (DT) algorithm by applying tolerance to it. We also discuss our time series reconstruction algorithm to recover the original granularity from a reduced dataset. Lastly, we talk about how to combine all the techniques into a processing pipeline to effectively manage time series data.

1) Overview: Figure 1 depicts the position of our techniques within a standard time series data management system. In this system, streaming monitoring data is first sent to a buffer before being processed and stored in a database. The buffer is designed to hold a chunk of data points, which undergoes tolerance calculation and deduplication. Specifically, the *Tolerance Calculation* component determines the metric-based tolerance, deciding which readings to keep or discard in the buffer chunk. Meanwhile, the *Deduplication* component removes redundant readings from the time series data and stores the remaining data points in the time series database, grouped by data schema and table. Alternatively, these components can process historical data to eliminate duplicated data points. To retrieve data, the *Reconstruction* component acts as a proxy for the real query to the time series database, reconstructing data points based on user queries. It's worth noting that data volume can be further reduced through aggregation and compression techniques. However, as these techniques are widely used in database management, we will not be discussing them in detail in this study.

Algorithm 1 Deduplication Simple (DS)

Input: $records$ array
Output: $deduplicated$ array

1: $previous \leftarrow$ stores node-label readings
2: **for** $i = 0, 1, \ldots,$ length of $records$ **do**
3: extract $node$, $label$, and $value$ from $records[i]$
4: **if** $previous[node][label]$ is empty or $value \neq previous[node][label]$ **then**
5: $previous[node][label] \leftarrow value$
6: append $records[i]$ to $deduplicated$
7: **end if**
8: **end for**

2) Deduplication Simple: The *Deduplication Simple* (DS) algorithm receives as input a `records` array from a given metric sorted by timestamp in ascending order. We first initialize a dictionary variable called `previous` which stores the previous readings of each combination of node and label (i.e. the readings from the same sensor). We refer to this combination as "node-label" moving forward. Next, we start iterating through the entire records array. In the first instance at time *T*, the `previous` of each node-label will be empty; therefore, we simply assign the first value reading to `previous` of that node-label and append the record to our `deduplicated` array. Moving forward, when we encounter the same node-label combination at time *T+1, T+2, …, T+k*, where *k* is the integer reflecting the length of unique timestamps. We compare the current reading to the one stored at `previous` for the given node-label. If the values are the same, we simply ignore the current reading; otherwise, we update `previous` and append the record to our `deduplicated` array. Once we are done iterating through the entire records array, we return its subset `deduplicated` array. The pseudocode for DS is given in Algorithm 1.

3) Tolerance Calculation: In the DS algorithm, we only ignore the current readings of each node-label that are identical to their previous reading, deduplicating exactly the same readings. However, to further reduce volume, we may add a metric-based tolerance window that deduplicates readings falling within it. The *Tolerance Calculation* (TC) algorithm receives as input the `records` array from a given metric and the `formula` selected to calculate the tolerance. As shown in Algorithm 2, we first initialize a dictionary variable called `readings`. Next, we start iterating through the entire `records` array to save the readings in separate arrays for each node-label in the `readings` dictionary. Once they are clustered, we pass each reading array for each node-label to the `formula` to calculate its tolerance and store each in the `tolerances` dictionary that gets returned.

4) Tolerance Formulas: To calculate the tolerance, we need a formula that relates to the dataset's nature and produces a relatively small number in magnitude; otherwise, it might create a tolerance window so large that could cover every reading.

First, we used the standard deviation as our tolerance

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

Algorithm 2 Tolerance Calculation (TC)

Input: $records$ array, $formula$ function
Output: $tolerances$ map

$readings \leftarrow$ stores arrays of node-label readings
for $i = 0, 1, \ldots,$ length of $records$ **do**
    extract $node$, $label$, and $value$ from $records[i]$
    append $value$ to $readings[node][label]$
end for
for $i = 0, 1, \ldots,$ nodes in $readings$ **do**
    for $j = 0, 1, \ldots,$ labels in $readings[i]$ **do**
        $tolerances[i][j] \leftarrow formula(readings[i][j])$
    end for
end for

formula since it directly relates to the dataset and it produces a relatively small value. However, even if the standard deviation of the dataset is low, we found that it is still not small enough to subtract and add to reading values to create a feasible tolerance window. Therefore, we compute the square root of the standard deviation to reduce its magnitude:

$$Tolerance = \sqrt{\sigma} = \sqrt{\sqrt{\frac{\Sigma_{i=1}^{n}(x_i - \mu)^2}{n}}} \tag{1}$$

where σ is the standard deviation of the sensor readings within a time window, x_i represents each reading, μ is the average reading, and n is the total number of readings of the sensor. In addition, we crafted another tolerance formula using the square root of the mean from the dataset, as shown in formula 2.

$$Tolerance = \sqrt{\mu} = \sqrt{\frac{\Sigma_{i=1}^{n} x_i}{n}} \tag{2}$$

Both formula 1 and formula 2 return a relatively low value in magnitude that is related to the dataset and that we can subtract and add to a previous reading to create the tolerance window.

Lastly, we combined σ and μ and calculated the coefficient of variation (CV) as the tolerance formula:

$$Tolerance = \frac{\sigma}{\mu} = \frac{\sqrt{\frac{\Sigma_{i=1}^{n}(x_i - \mu)^2}{n}}}{\frac{\Sigma_{i=1}^{n} x_i}{n}} \tag{3}$$

However, the value calculated from this formula returns a result between 0 and 1 because the standard deviation is often smaller than the mean. Consequently, adding and subtracting the tolerance to the previous reading, as we did with the other two formulas, would result in a tolerance window that is too narrow. To create a suitable tolerance window, we multiply the tolerance by the reading, then add and subtract the resulting value from the reading.

5) Deduplication & Tolerance: We are now able to expand DS into the *Deduplication & Tolerance* (DT) algorithm. The tolerance window is calculated by buckets for each node-label. For example, if we are processing a historical data span of one day, we may set the bucket size to one hour. Deduplication and tolerance calculations are then applied to each one-hour interval. The pseudocode for DT is provided in Algorithm 3. In this algorithm, the `records` variable holds the whole dataset to be processed, `formula` represents the tolerance formula, and `gap` denotes the bucket size in terms of time.

Algorithm 3 Deduplication & Tolerance (DT)

Input: $records$ array, $formula$ function, gap hours
Output: $deduplicated$ array

$previous \leftarrow$ stores node-label readings
$start \leftarrow$ first $records$ timestamp
$finish \leftarrow start + gap$
while True **do**
    $bucket \leftarrow records[start, finish]$
    if length of $bucket \leq 0$ **then**
        finished processing, exit...
    end if
    $tolerances \leftarrow TC(bucket, formula)$
    for $i = 0, 1, \ldots,$ length of $bucket$ **do**
        if $previous$ is empty or
         $value < previous - tolerances$ or
         $value > previous + tolerances$ **then**
            $previous \leftarrow value$
            append $bucket[i]$ to $deduplicated$
        end if
    end for
    $start \leftarrow finish$
    $finish \leftarrow finish + gap$
end while

6) Reconstruction: The reconstruction process begins by querying the deduplicated data from the time-scale database, based on user-specified criteria such as time range and node-label. Similar to the *Deduplication & Tolerance* (DT) process discussed earlier, the deduplicated data is organized into buckets according to the time interval. Within each bucket, if a timestamp is missing, we simply repeat the previous value since the value corresponding to that timestamp has been deduplicated by the tolerance window. It is important to note that the data collection frequency is known, allowing us to easily identify which time-value pairs have been discarded.

7) Processing Pipeline: To maximize the effectiveness of the techniques, we recommend using them in combination. Therefore, we propose a processing pipeline for managing time series data. The first step is to determine whether to reduce historical data or handle streaming data. If historical data reduction is chosen, we need to establish a threshold for triggering the deduplication of time series data. For example, we can execute DS/DT on tables that have timestamps older than 7 days from the current time. On the other hand, if streaming data deduplication is desired, we should apply DS/DT before inserting records into the database. Additionally, it is important to define the buffer size, such as grouping 1 hour's worth of time series data for deduplication prior to insertion.

Our deduplication techniques complement existing data volume reduction methods like aggregation and compression,

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

commonly employed in time-series databases. By combining these techniques, a substantial reduction in data volume can be achieved while preserving data granularity to the greatest extent possible. The deduplication approach described in this paper can be implemented as a data management policy. For instance, we could establish a policy to deduplicate historical records older than 30 days. Alternatively, the deduplication policy could be applied to aggregated data to further reduce its volume. These flexible policies offer valuable options for optimizing storage and maintaining efficient data management in various scenarios.

IV. Experimental Results

In this Section, we demonstrate the deduplication rates achieved while executing the DS and DT algorithms, reconstruction errors, and query runtime for the techniques proposed.

A. Deduplication Rates

In Figure 2, we present the deduplication rates (volume reduction) achieved by increasing the data volume for both the DS algorithm and DT with our three formulas, across different metrics. A notable observation is the significant variation in deduplication rates for the same technique across different metric types. Among the formulas used, the *Coefficient of Variation (CV)* (Algorithm 3) produced the most consistent deduplication results across various tables and reduce the data volume by 70.38% on average.

Furthermore, for the *RPM Reading* data, we observed that the *Square Root of Standard Deviation* formula (Algorithm 1) resulted in a tolerance window that was so small that it performed similarly to the DS algorithm. This observation suggests that the standard deviation for that particular dataset is very low and may not be suitable for tolerance calculation. However, this observation does not hold true for other datasets, as the same formula achieved significantly higher deduplication rates compared to DS. It is worth noting that only for the *RPM Reading* data did the CV formula outperform the *Square Root of Mean* formula (Algorithm 2).

Lastly, we observed that there is no direct relationship between the deduplication rate and the increasing data volume. Instead, the deduplication rate is determined by the variability of values within the dataset, which causes the values to fall within or outside of the tolerance window with varying frequency.

B. Reconstruction Error

In addition to analyzing volume reduction, it is equally important to evaluate the accuracy of the reconstruction by examining the errors. If the reconstructed records deviate too much from the original data, the analysis would yield incorrect conclusions. Therefore, we computed the *Mean Average Percentage Error* (MAPE) between the original and reconstructed records for each metric after applying DT with the CV formula. To ensure accurate comparisons, we performed the MAPE calculation for each node-label within each metric.

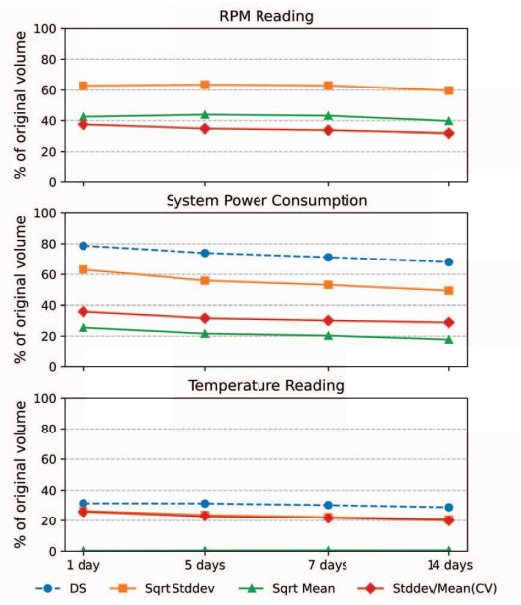

Figure 2. Volume reduction using different tolerance functions. In the RPM Reading dataset, the DS line and the Sqrt Stddev line are overlapped.

Subsequently, we present the maximum and average MAPE values across all node-labels in Figure 3.

Upon analysis, we observed that, in some instances, the error for *System Power Consumption* was relatively higher compared to the other two metrics. However, the average MAPE values for all three metrics exhibited consistently low and closely uniform error rates (0.74% on average). This indicates that the reconstruction process, utilizing the CV formula, generally resulted in accurate and reliable records, with minimal divergence from the original dataset.

C. Query Performance

As per our processing pipeline proposal, we conducted performance measurements to compare the query times of the original records (without any technique applied) against the query times of tables containing only deduplicated records, along with the additional runtime required for reconstruction. The results, depicted in Figure 4, illustrate the performance across different metrics with increasing data volumes.

As depicted in the figure, the data fetch time demonstrates a substantial reduction for all metrics across various time ranges, thanks to the reduced data volume achieved through our proposed deduplication techniques. However, it is important to note that the reconstruction process, aimed at restoring the original data granularity, introduces additional runtime overhead, particularly for longer data ranges. In contrast, for shorter time ranges and certain metrics (such as the Temperature Reading metrics), the total overhead is negligible.

V. Related Works

A comprehensive review of the literature reveals a wide-ranging perspective on the management and analysis of tempo-

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

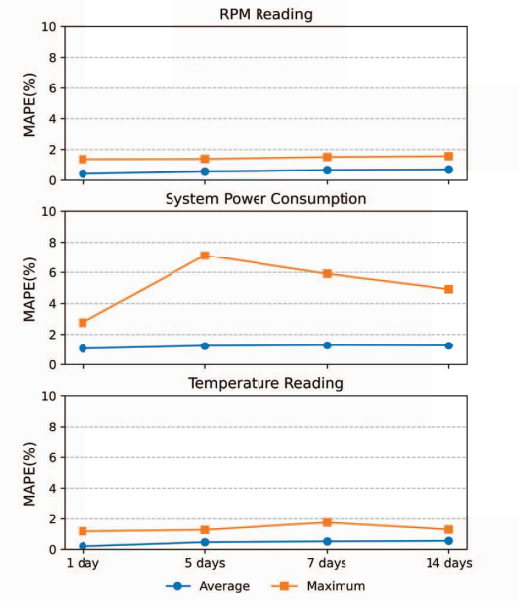

Figure 3. Reconstruction error using the Coefficient of Variation (CV) tolerance formula.

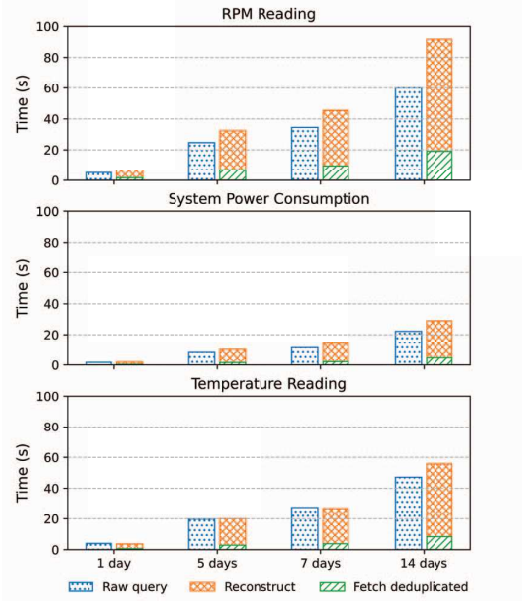

Figure 4. Runtime in seconds for deduplicated data fetching & reconstruction.

ral big data. Several state-of-the-art methodologies addressing various aspects, such as modeling, mining, indexing, storage, compression, and querying, are covered in surveys [10]–[13]. Furthermore, there exist several surveys specifically dedicated to the processing and analysis of streaming data [14]–[17].

The need for efficient archiving of hierarchical data is addressed in Wang et al. [18]. The authors propose an XML archiving system that combines compact data and timestamp storage with optimization techniques for evaluating queries with temporal constraints. Vestergaard et al. instantiate a generalized deduplication approach to introduce a novel principle for on-the-fly compression of large volumes of time series data [19]. Gil et al. [20] present improvements to index construction for distortion-free subsequence matching in time series databases using dynamic programming techniques. The work by Yu et al. [21] introduces a cloud-computing approach for managing peta- and exa-scale time series data from large sensor networks. They leverage well-established data storage and processing paradigms such as Bigtable [22] and MapReduce [23].

The foundational aspects of data stream management are discussed in Golab et al. [24]. The paper addresses various aspects including application requirements, data models, continuous query languages, etc. Liu et al. [14] present a comprehensive survey of existing real-time processing systems. These systems offer advantages in handling uninterrupted data streams and enable real-time data analytics. Bai et al. [25] introduce a time stamp memory and a mechanism for generating enabling time stamps on demand. This approach effectively reduces both memory usage and query latency in data stream management systems that involve union and join operations.

VI. Conclusions & Future Directions

In this paper, we investigated the techniques of Deduplication Simple (DS), metric-based Tolerance Calculation (TC), Deduplication & Tolerance (DT), and Reconstruction to reduce the volume of time series databases while preserving fine granularity. Additionally, we proposed a processing pipeline that effectively manages time series data. The benchmarks conducted in this study demonstrated the efficacy of these volume reduction techniques in eliminating redundant time series data, while maintaining low reconstruction errors and minimal query runtime overhead.

Several avenues for future research and development are worth considering. Firstly, exploring different tolerance calculation formulas could enhance the adaptability of the deduplication and tolerance calculation techniques. Furthermore, applying these techniques to diverse datasets would provide valuable insights into their generalizability and potential improvements. Additionally, investigating the potential of parallelization to optimize the deduplication and reconstruction processes holds promise for achieving even greater efficiency in time series data management.

VII. Acknowledgement

We would like to express our gratitude to the anonymous reviewers for their insightful comments and suggestions. This research is supported in part by the National Science Foundation under grants OAC-1835892, CNS-1817094, and CNS-1939140. We are also very grateful to the High-Performance Computing Center of Texas Tech University for providing HPC resources for this project.

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Y. Katsis, Y. Freund, and Y. Papakonstantinou, "Combining databases and signal processing in plato." in *CIDR*, 2015.

[2] J. Ronkainen and A. Iivari, "Designing a data management pipeline for pervasive sensor communication systems," *Procedia Computer Science*, vol. 56, pp. 183–188, 2015.

[3] HPCC. (2021) High Performance Computing Center. [Online]. Available: http:www.depts.ttu.edu/hpcc/

[4] TimescaleDB. (2023) TimescaleDB. [Online]. Available: https://www.timescale.com/

[5] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[6] Z. Xue, H. Qian, L. Shen, and X. Wu, "A comprehensive study of present data deduplication," in *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. IEEE, 2021, pp. 1748–1754.

[7] A. Venish and K. Siva Sankar, "Study of chunking algorithm in data deduplication," in *Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015, Volume 2*. Springer, 2016, pp. 13–20.

[8] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou, "Big data challenge: a data management perspective," *Frontiers of computer Science*, vol. 7, pp. 157–164, 2013.

[9] J. Li, G. Ali, N. Nguyen, J. Hass, A. Sill, T. Dang, and Y. Chen, "Monster: an out-of-the-box monitoring tool for high performance computing systems," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2020, pp. 119–129.

[10] A. Cuzzocrea, "Temporal aspects of big data management: state-of-the-art analysis and future research directions," in *2015 22nd International Symposium on Temporal Representation and Reasoning (TIME)*. IEEE, 2015, pp. 180–185.

[11] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2581–2600, 2017.

[12] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data*, vol. 6, no. 1, pp. 1–37, 2019.

[13] G. Chiarot and C. Silvestri, "Time series compression survey," *ACM Computing Surveys*, vol. 55, no. 10, pp. 1–32, 2023.

[14] X. Liu, N. Iftikhar, and X. Xie, "Survey of real-time processing systems for big data," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 356–361.

[15] M. Ghesmoune, M. Lebbah, and H. Azzag, "State-of-the-art on clustering data streams," *Big Data Analytics*, vol. 1, pp. 1–27, 2016.

[16] B. Yadranjiaghdam, N. Pool, and N. Tabrizi, "A survey on real-time big data analytics: applications and tools," in *2016 international conference on computational science and computational intelligence (CSCI)*. IEEE, 2016, pp. 404–409.

[17] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, pp. 154 300–154 316, 2019.

[18] H. Wang, R. Liu, D. Theodoratos, and X. Wu, "Efficient storage and temporal query evaluation in hierarchical data archiving systems," in *Scientific and Statistical Database Management: 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings 23*. Springer, 2011, pp. 109–128.

[19] R. Vestergaard, Q. Zhang, and D. E. Lucani, "Lossless compression of time series data with generalized deduplication," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[20] M.-S. Gil, B.-S. Kim, M.-J. Choi, and Y.-S. Moon, "Fast index construction for distortion-free subsequence matching in time-series databases," in *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*. IEEE, 2015, pp. 130–135.

[21] B. Yu, A. Cuzzocrea, D. Jeong, and S. Maydebura, "On managing very large sensor-network data using bigtable," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 2012, pp. 918–922.

[22] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[23] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[24] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM Sigmod Record*, vol. 32, no. 2, pp. 5–14, 2003.

[25] Y. Bai, H. Thakkar, H. Wang, and C. Zaniolo, "Time-stamp management and query execution in data stream management systems," *IEEE Internet Computing*, vol. 12, no. 6, pp. 13–21, 2008.

Authorized licensed use limited to: Texas Tech University. Downloaded on January 04,2025 at 17:39:39 UTC from IEEE Xplore. Restrictions apply.