TV Store: Automatically Bounding Time Series Storage via Time-Varying Compression

Yanzhe An, Yue Su, Yuqing Zhu, Jianmin Wang

Speaker wrl

FAST 2022

Background

➢ **Time-series Database**: The time series database is mainly used to process data with time tags (changing in time order, namely time serialization), and is mainly used for the prediction and analysis of long-term events.

metric → Wind

timestamp	direction	speed	sensor	city
1467627245000	23.4	3.4	95D8-7913	上海
1467627245000	145.1	1.1	F3CC-20F3	北京
1467627247000	23.2	3.3	95D8-7913	上海
1467627247000	145.0	1.2	F3CC-20F3	北京
1467627255000	23.3	3.3	85D8-7913	上海

Data point

field

tag

➢ **Trend & Big data**: Time series databases are becoming more and more popular, but the demand for data storage is becoming more and more serious.

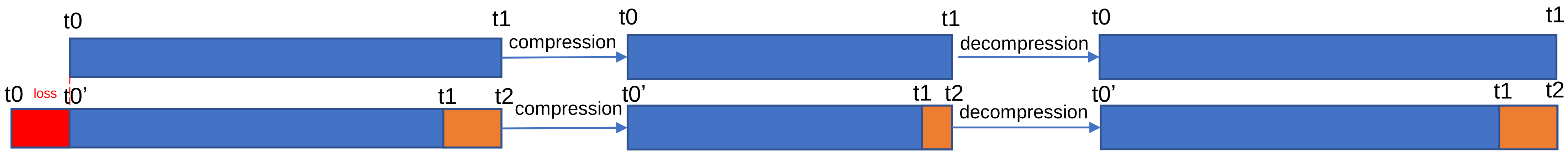
- balance between data volume and storage cost
- storage space is restricted in some specific deployments
- Data compression is demanded for reducing data in both transmission and storage

➢ **An effective storage management strategy that can constrain the storage space is desirable and important for time-series databases.**

Background

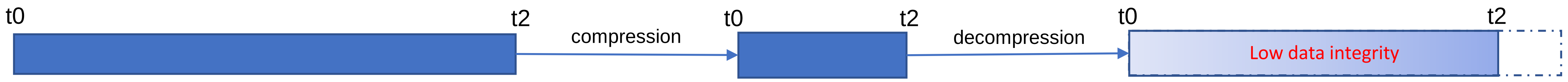
➢ **Lossless Compression**:

- The original data can be completely recovered without any distortion
- Lower upper limit of achievable compression ratio
- Time series databases commonly control storage consumption by directly discarding data older than a given time or exceeding a storage threshold. （Short data time span）



➢ **Lossy compression**:

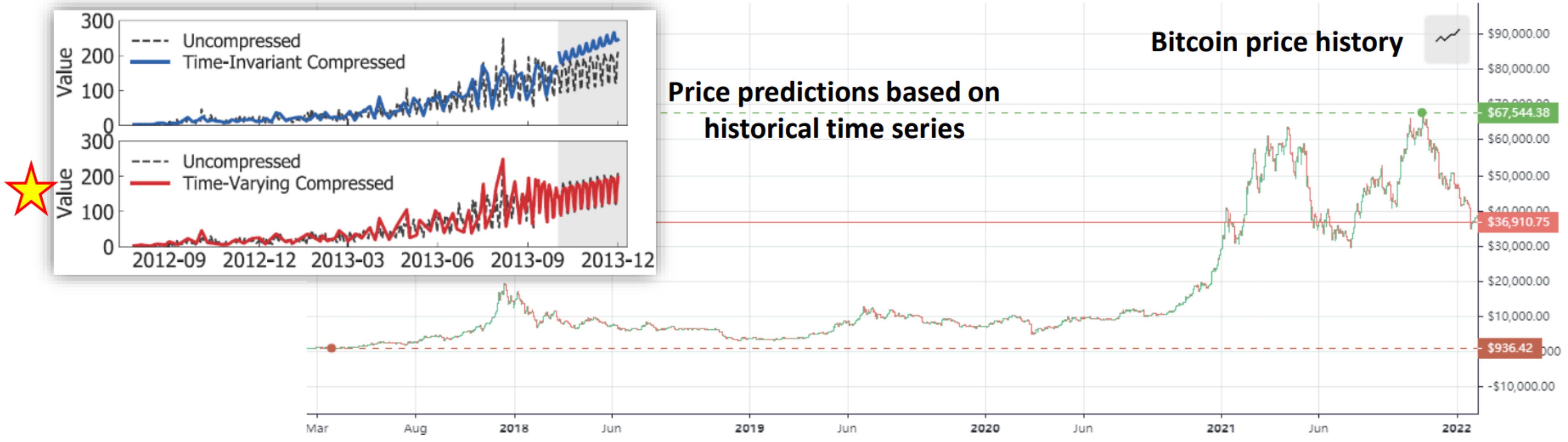
- Decompression can only restore partial data
- Higher upper limit of achievable compression ratio.
- Incomplete or inaccurate data recovery will affect the results of subsequent analysis or prediction.(Low data integrity)



Problem & Motivation

- The problem: In the time-series database storage, **the contradiction between the time span of data and the integrity of data.**
- The motivation: **The importance of time series data changes along with time.**

Faster and more accurate predictions on the same small volume of data



Idea&Challenge

- **Key Idea:** **time series data can be compressed losslessly or lossily according to its importance, which is in turn related to its age.**

  - The information of new data will be more important, and users will not be able to accept the loss of new data. ——**lossless compression**
  - Users commonly accept information loss on less important old data. ——**lossily compression**

- **Method:** **A compression method that can automatically adjust according to time series and meet the specified compression rate.**

  - **challenge 1 :** How to **efficiently** compress data according to time series attributes, as data keep being ingested?

  - **challenge 2 :** How to ensure that the data size can be consistently maintained within the specified compression threshold？

Design —Time-Varying Compression

- **Key question 1: How to select the appropriate compression ratio for time series data**

- **Key idea: Fitting with a non-decreasing function varying with time**

r1=r(1)=2   m=5
r2=r(2)=3   ra =2

- data for compression is kept in the unit of chunks

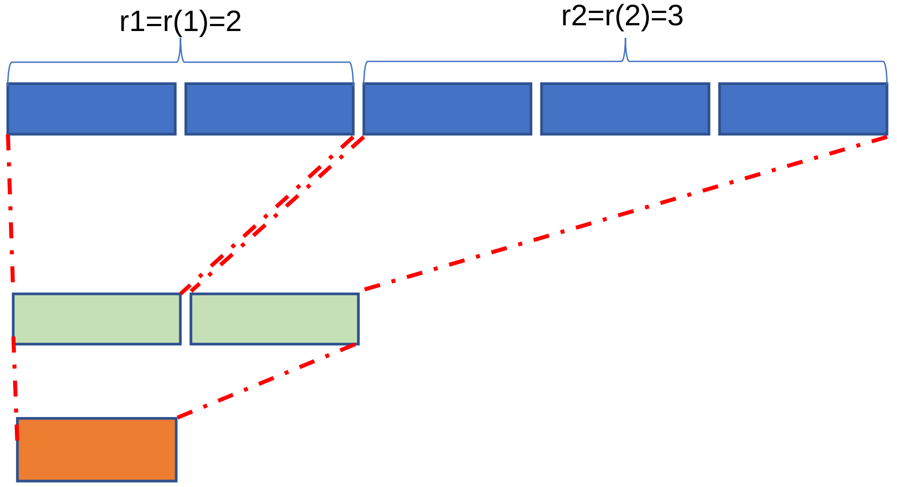
- Compression ratio sequence generated by function r(t)

  r1 r2 r3 r4 … rk

- Ratio compliance by approximation

$$\Sigma_i^k r_i \geq m \tag{1}$$

$$m/k \geq \bar{r} \tag{2}$$

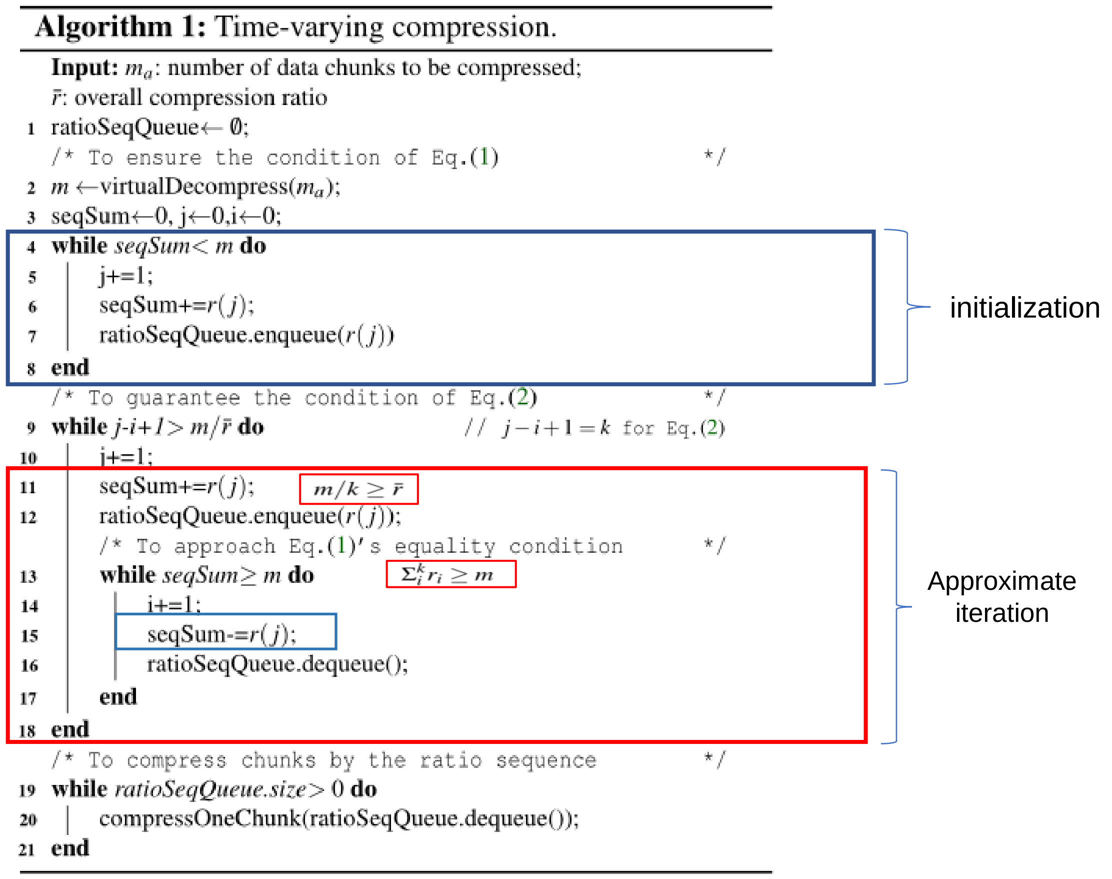


- For new data, we compress them losslessly or with a low ratio by lossy compression.
- For unimportant data, we compress them by a high compression ratio by lossy compression.

Design —Time-Varying Compression

➢ **Key question 2: How to quickly recalculate the compression ratio sequence with new data?**

➢ **Key idea: Through virtual decompression technology**

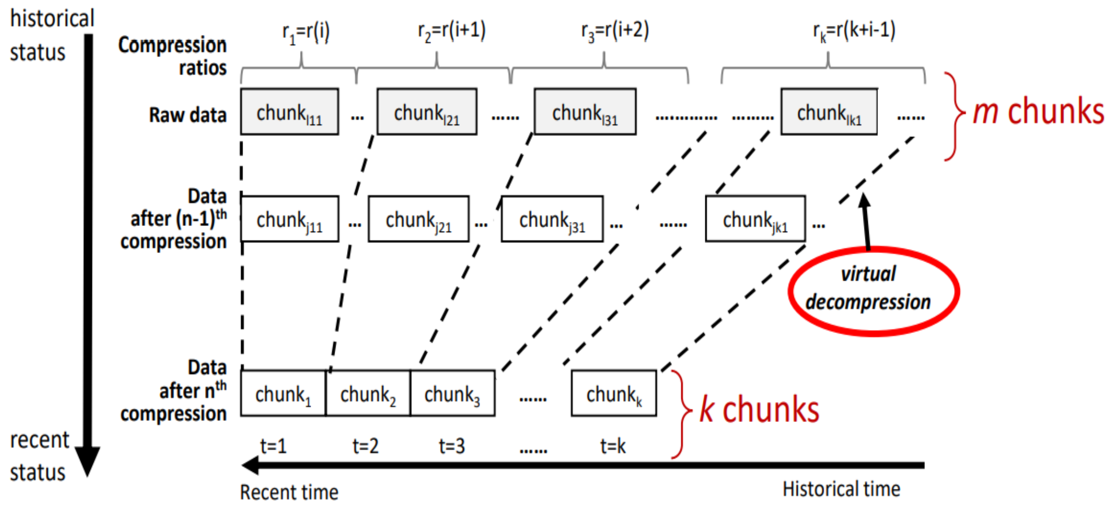


Algorithm 1: Time-varying compression.

```
Input: $m_a$: number of data chunks to be compressed;
       $\bar{r}$: overall compression ratio
ratioSeqQueue ← ∅;
/* To ensure the condition of Eq.(1)                    */
$m$ ← virtualDecompress($m_a$);
seqSum←0, j←0,i←0;
while $seqSum < m$ do
    j+=1;
    seqSum+=$r(j)$;
    ratioSeqQueue.enqueue($r(j)$)
end
/* To guarantee the condition of Eq.(2)                 */
while $j$-$i$+$1 > m/\bar{r}$ do          // $j-i+1=k$ for Eq.(2)
    j+=1;
    seqSum+=$r(j)$;    $m/k \geq \bar{r}$
    ratioSeqQueue.enqueue($r(j)$);
    /* To approach Eq.(1)'s equality condition           */
    while $seqSum \geq m$ do    $\Sigma_i^k r_i \geq m$
        i+=1;
        seqSum-=$r(j)$;
        ratioSeqQueue.dequeue();
    end
end
/* To compress chunks by the ratio sequence             */
while $ratioSeqQueue.size > 0$ do
    compressOneChunk(ratioSeqQueue.dequeue());
end
```

initialization

Approximate iteration



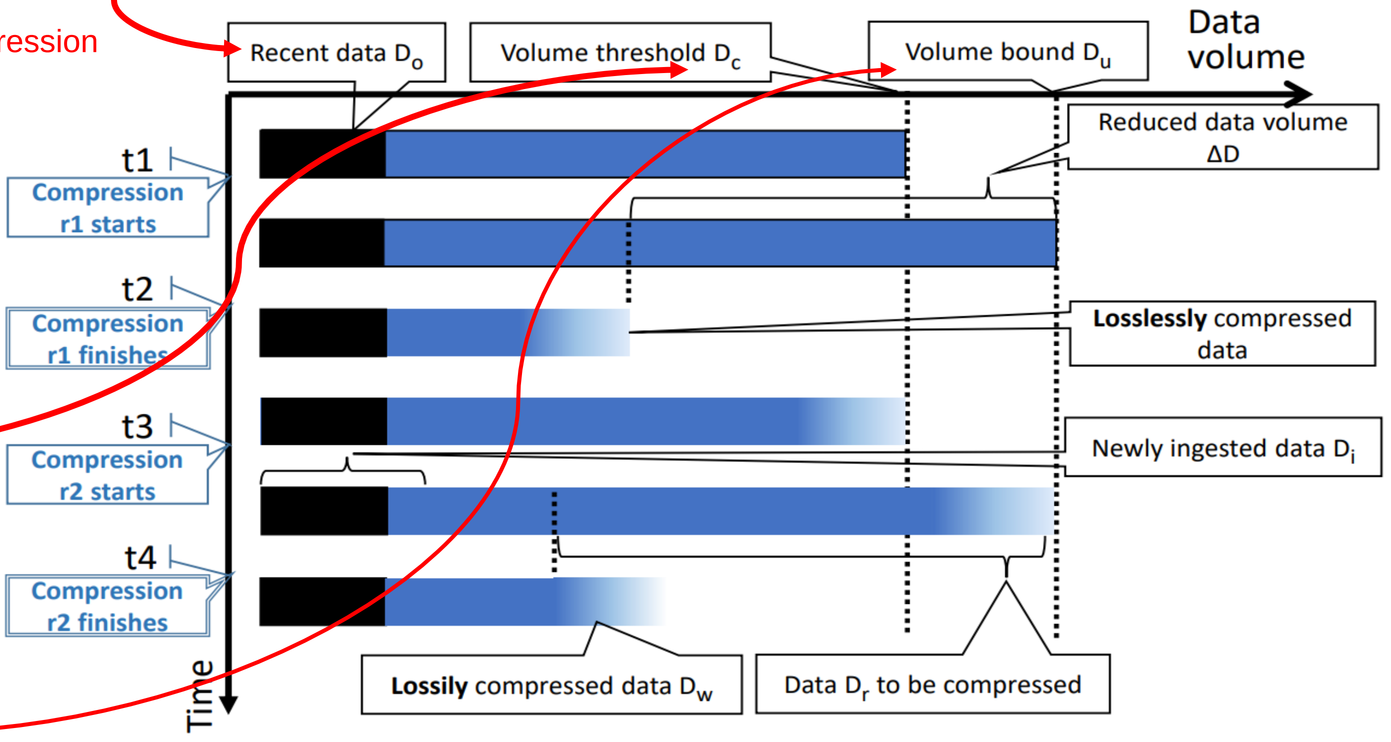
Design —Automatic bounding

- **Key question 1: Whether to compress all data?**
  - Latest data: The latest data should be used directly without compression

- **Key question 2: What ratio to compress?**
  - Too large: losing information unnecessarily
  - Too small: exceeding storage bound

- **Key question 3: When to compress?**
  - Too early: losing information unnecessarily and involving unnecessary costs
  - Too late: exceeding storage bound

- **Key question 4: Whether to save all data?**
  - oldest: Very old data is damaged due to lossy compression, and its availability is low



Design —Automatic bounding

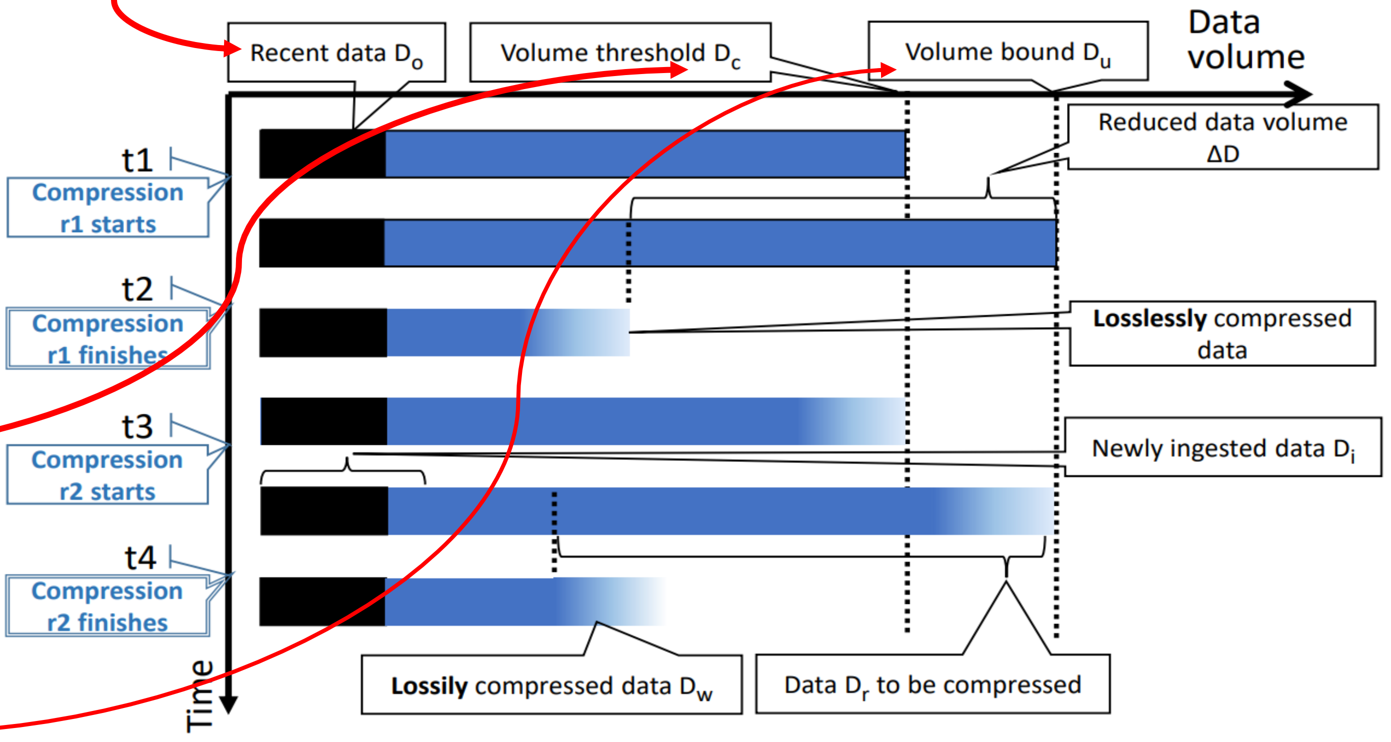
- **Key question 1: Whether to compress all data?**
  - Solution 1: Users can set the size to be used. (Do)
- **Key question 2: What ratio to compress?**
  - Solution 2:

$$r_c \geq \frac{v_r}{v_r - v_i}$$

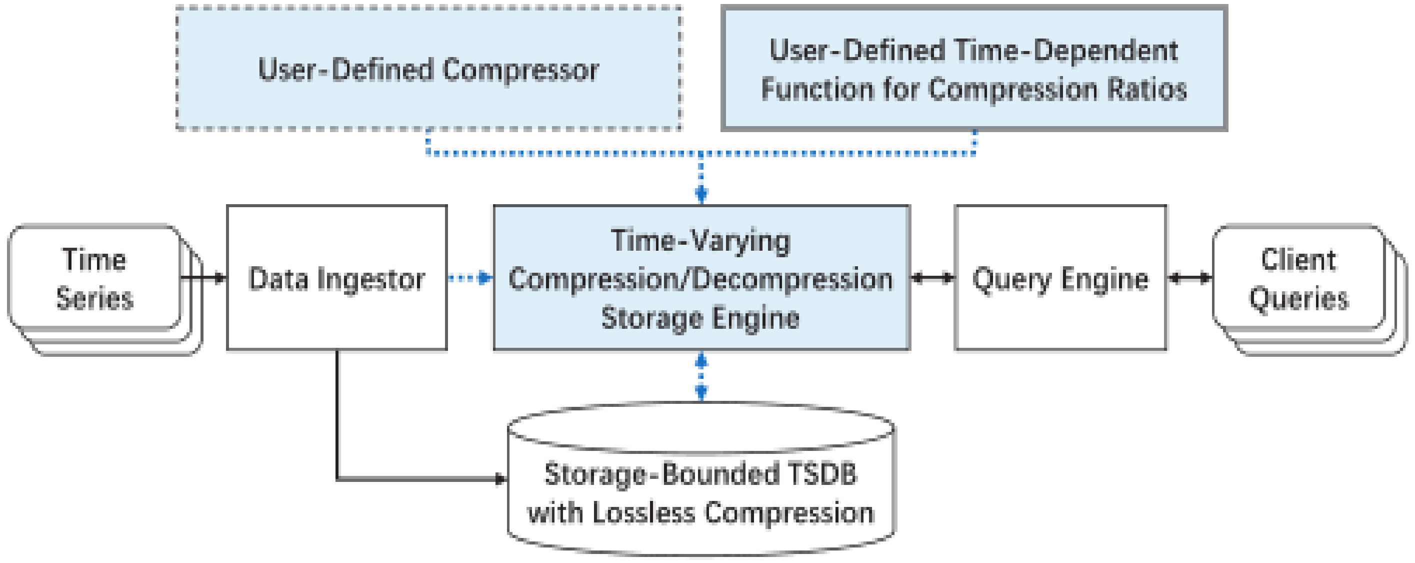
- **Key question 3: When to compress?**
  - Solution 3:

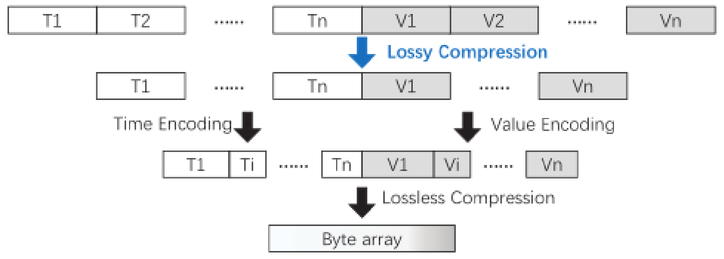
$$D_c \leq (D_u - D_o)/(\frac{v_i}{v_r} + \frac{1}{r} + 1) + D_o$$

- **Key question 4: Whether to save all data?**
  - Solution 4: Allows users to specify a compression ratio rmax or an error rate emax



Architecture


User-Defined Compressor
User-Defined Time-Dependent Function for Compression Ratios
Time Series
Data Ingestor
Time-Varying Compression/Decompression Storage Engine
Query Engine
Client Queries
Storage-Bounded TSDB with Lossless Compression



PLA Compressor(default)

- While exponential, power-law and constant functions are all supported as the time-dependent ratio functions, TVStore uses the power-law function as the default.

- TVStore allows users to set the upper bound of storage space and the largest compression ratio permitted.

Evaluate

- Datasets

Real-world datasets	REDD public dataset **(7.5TB)**	Train-load private dataset **(6.6TB)**	
Synthetic datasets	Uniform random **(5TB)**	Poisson distribution **(5TB)**	Pareto distribution **(5TB)**

- Workloads

Ingestion	With compression ratios at 1X, 20X, 60X, 100X
Query	Aggregations (sum, avg, max, min) for data at Age(*S*) with Length(*S*), *S*=(Mon/Millennia, Day/Century, Min/Recent)

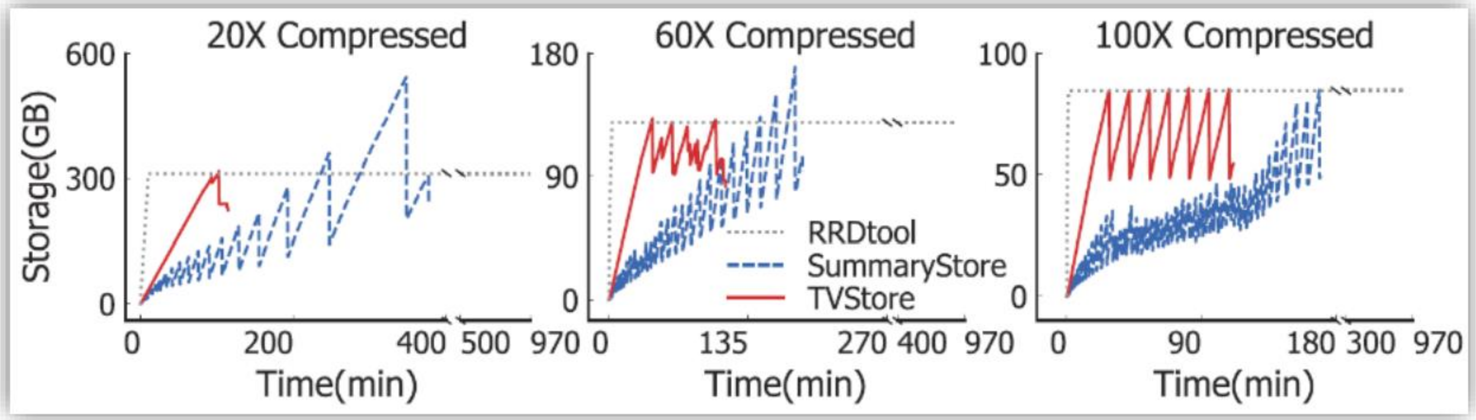
- Compared systems

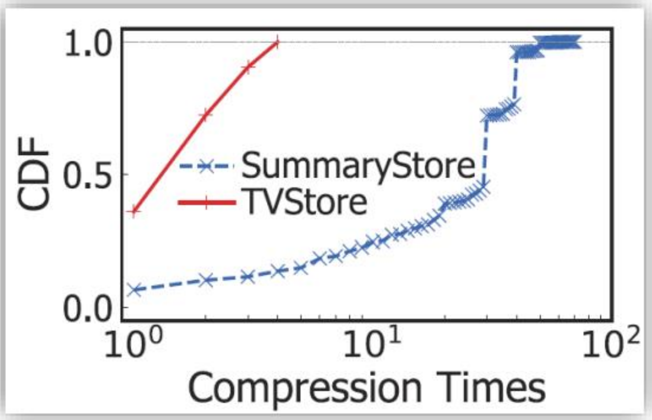
SummaryStore	**RRDTool**	**Apache IoTDB**
Approximate time series store	Round-robin time series DB	Implementation baseline

- Hardware instances:
  - Setting 1: two Intel Xeon E5-2650 CPUs, 370GB DDR4 memory
  - Setting 2: 32GB memory and an 8-core CPU

Evaluate —Storage bounding & compression cost

- **TVStore effectively bounds its storage with high ingestion performance.**
  - RRDTool bounds storage with low ingestion performance.
  - SummaryStore does not support storage bounding.



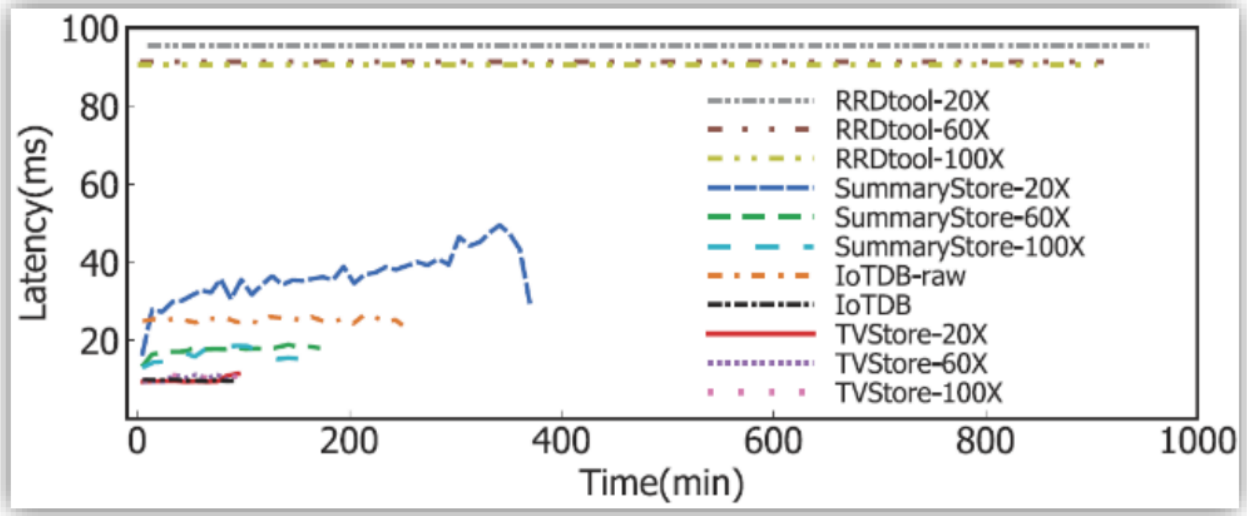


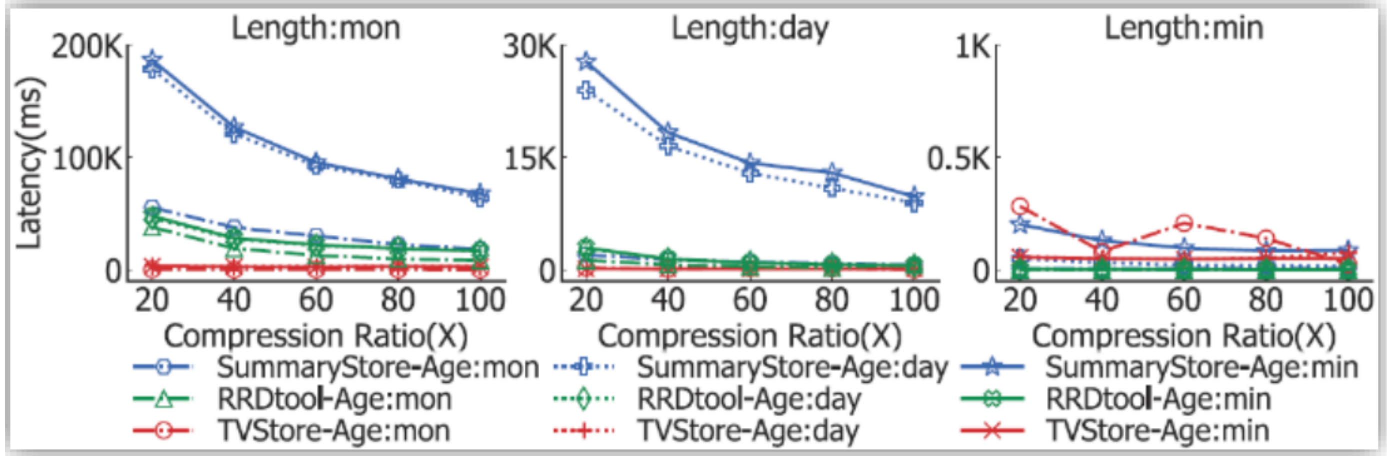
- **TVSTore requires fewer compression/merging times than SummaryStore.**
  - Incurring fewer disk I/Os and computation costs
  - Cold-data compression is more efficient than hot-data compression.

Evaluate —Ingestion & query performances

- TVStore has much **higher ingestion throughput** than SummaryStore and RRDtool in all cases.

- TVStore's compression process has **little impact** on the normal processing of writes.



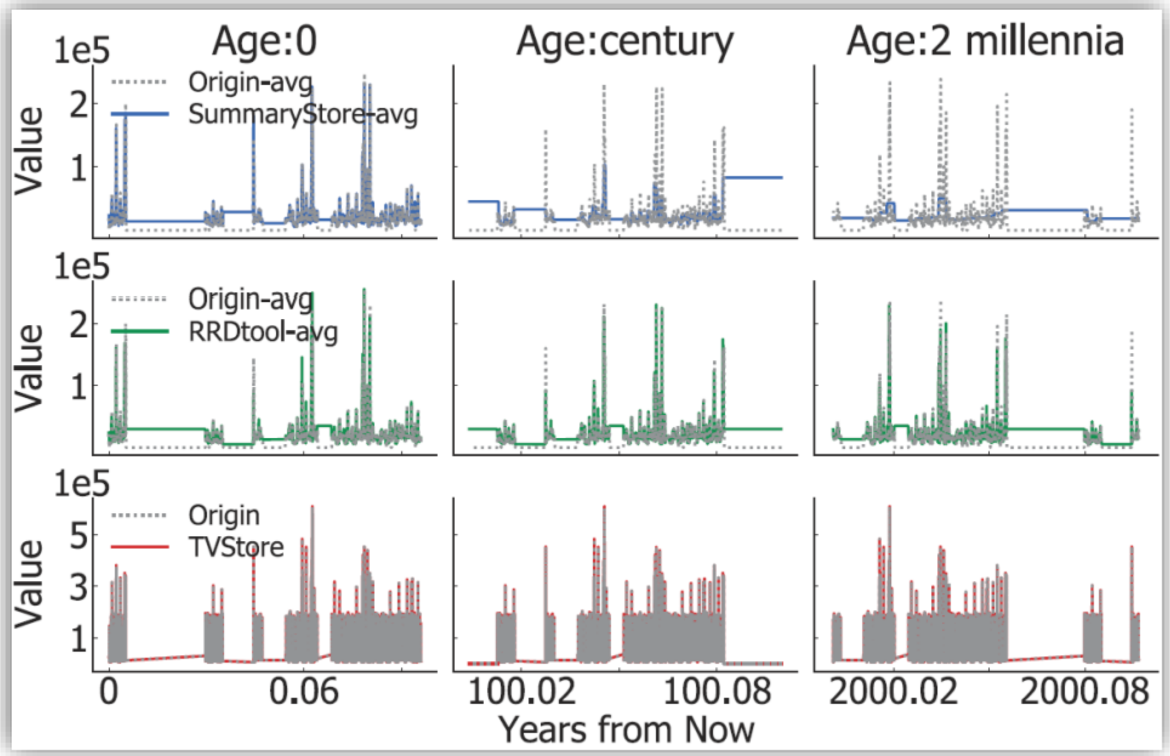


- TVStore implementation can answer queries **35X and 8.7X faster** than SummaryStore and RRDtool respectively for the best case.

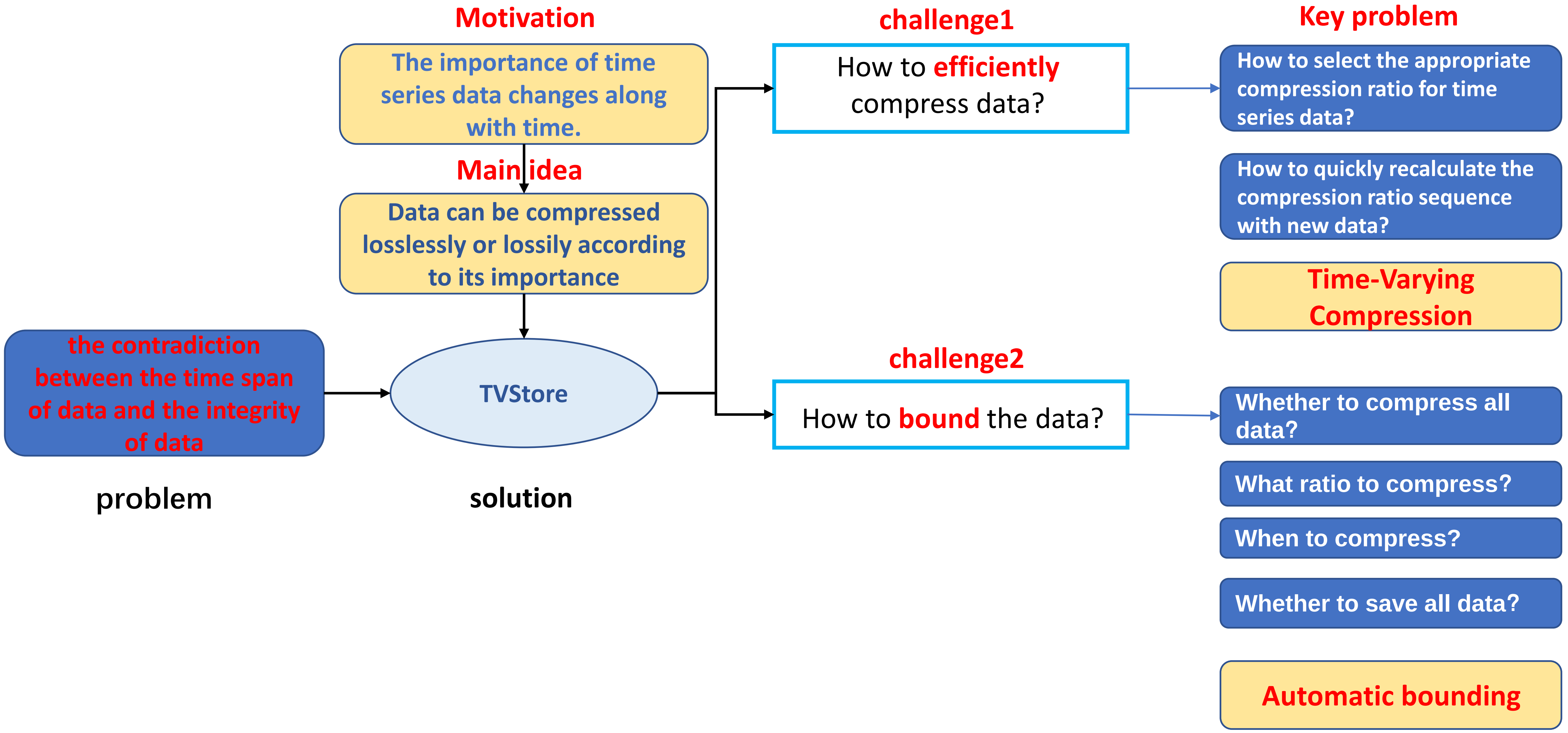
Evaluate —How data look in databases

- Time-varying pattern
  - TVStore and SummaryStore demonstrate **time-varying patterns**, while RRDtool has the time-invariant curves.

- **Preserving much more information**
  - Under the same overall data reduction/compression ratio, TVStore can restore data to almost **the same as the original**, while RRDtool and SummaryStore cannot.



Conclusion


Motivation
The importance of time series data changes along with time.
Main idea
Data can be compressed losslessly or lossily according to its importance
the contradiction between the time span of data and the integrity of data
TVStore
problem
solution
challenge1
How to efficiently compress data?
challenge2
How to bound the data?
Key problem
How to select the appropriate compression ratio for time series data?
How to quickly recalculate the compression ratio sequence with new data?
Time-Varying Compression
Whether to compress all data?
What ratio to compress?
When to compress?
Whether to save all data?
Automatic bounding