An Efficient Cloud Storage Model with Compacted Metadata Management for Performance Monitoring Timeseries Systems

FAST'26

Kai Zhang[1], Tianyu Wang[2] and Zili Shao[1]

[1]The Chinese University of Hong Kong, Hong Kong SAR, China
[2]Shenzhen University, Shenzhen, China

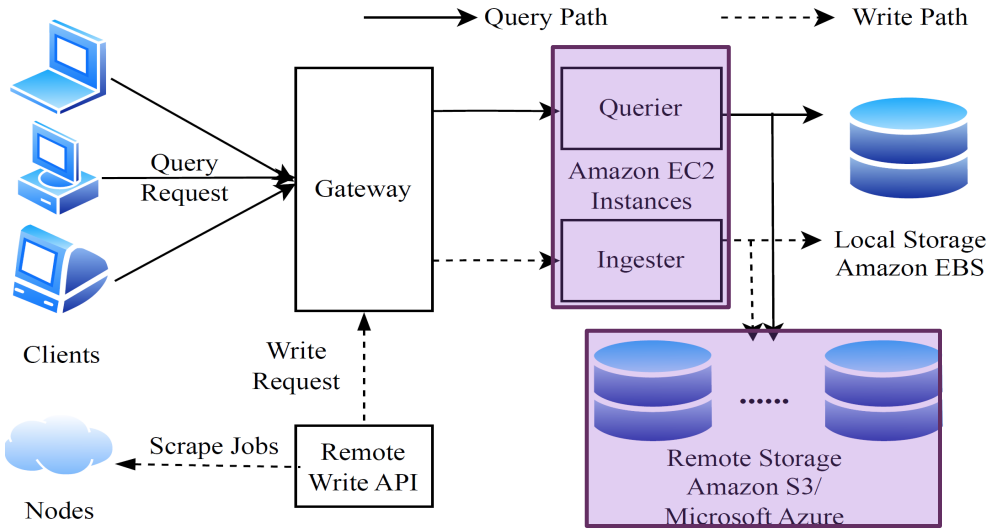
Outline

- Background
- Motivation
- CloudTS Architecture
- Experiments
- Conclusion

Performance Monitoring Timeseries Systems

- Timeseries Database (TSDB)
  - Collecting real-time performance metrics such as CPU utilization and memory usage
  - E.g., Prometheus via Cortex, TimescaleDB and InfluxDB

- Moving data to cloud object storage



Architecture of a cloud-based timeseries system.

Performance Monitoring Timeseries Systems

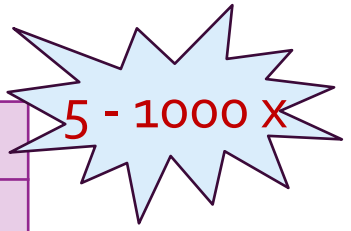
- Timeseries Database (TSDB)
  - Collecting real-time performance metrics such as CPU utilization and memory usage
  - E.g., Prometheus via Cortex, TimescaleDB and InfluxDB

- Moving data to cloud object storage
  - Advantages:
    - High scalability
    - Pay-as-you-go

- Disadvantages:
  - Prolonged access latency

Access Latency Comparison

Amazon S3	~20 - 50 ms
Local NVMe SSD	~10 - 100 µs
Local HDD	~2 - 10 ms



Performance Monitoring Timeseries

- A timeseries is defined by a set of tags, where each data point has a timestamp and a value.
- Each unique combination of tag pairs defines a distinct timeseries.
- Queried based on the combination of tag-based predicates and time range.

TS: {ts_metrics="cpu_usage", node="node01", cpu="core1"}

The monitored CPU usage information for CPU core "core1" on "node01".

Query example:

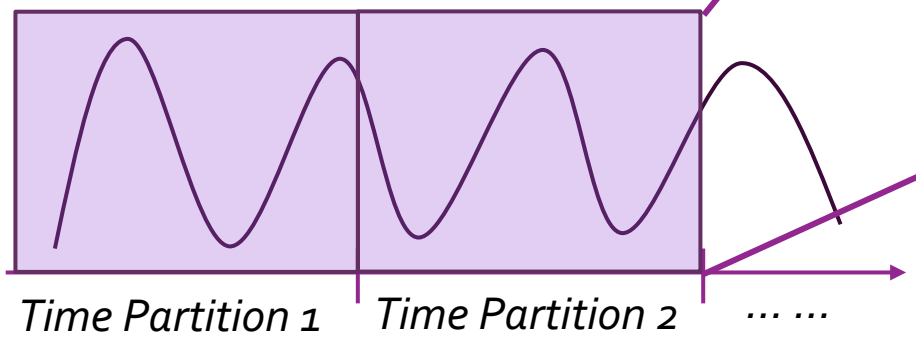
GET /api/v1/query_range?
query={ts_metrics="cpu_usage", node="node01"}&
start=2025-07-01T20:10:30.781Z&
end=2025-07-01T20:11:00.781Z

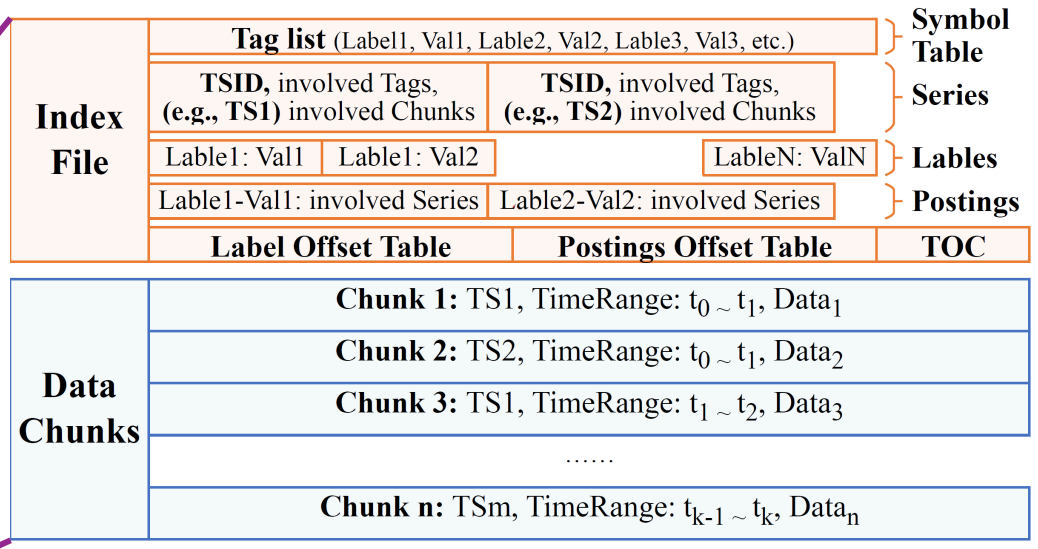
Tag pairs are used to define and identify timeseries.

Timeseries Data Organization

- Storage Models of Prometheus and Cortex
  - Tag-based timeseries indexing

    TS: {ts_metrics="cpu_usage", node="node01", cpu="core1"}
  - Time-partitioned data blocks
    - Metadata (Tags, inverted index)
    - Raw timeseries data chunks





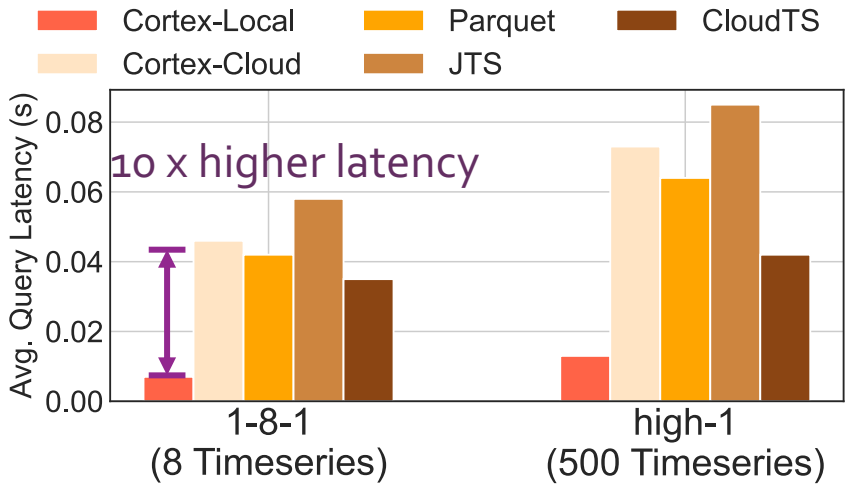
Data organization of Prometheus and Cortex (Data block in one time partition).

Outline

- Background
- Motivation
- CloudTS Architecture
- Experiments
- Conclusion

Key Issues in Cloud-based Timeseries Systems

- Compare average query latency among popular cloud-based timeseries storage approaches with local-based Cortex:
  - Traditional TSDBs
    - (e.g., Cortex, one data block -> one object)
  - Cloud-native formats
    - (e.g., Apache Parquet and JSON Time Series)



Goal

Issue 1: High Cloud Storage Latency
- A fundamental characteristic inherent to cloud object storage

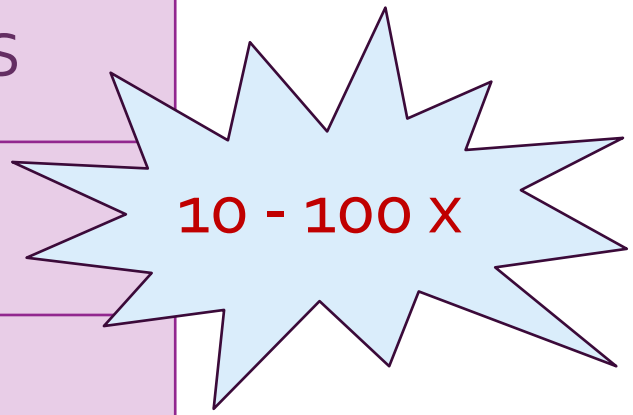
Issue 2: Severe Read Amplification
- Queries retrieve significantly more data than actually required.

Issue 3: Metadata Overhead
- High-cardinality queries incur severe overhead

Read Amplification in Cloud-based Timeseries Systems

Read amplification factor for query involving 10 TS in 1 hour

Cloud-Cortex	Block contains 1,000 TS -> Read 1,000 TS
Apache Parquet	Row group spans all TS -> Read all rows
JTS	Must scan all TS objects for tag filtering



Read amplification is a fundamental performance bottleneck.

- Unnecessary data are fetched for most queries
- I/O volume far exceeds the actual data needed

Timeseries Metadata Redundancy

- Timeseries Metadata Analysis

Dataset:
- 500K timeseries
- 200 time partitions

- Over 70% of tags are repeated across multiple timeseries
- Some tags (e.g., region) are shared by more than 90% of the timeseries

Metadata redundancy inflates metadata size and hampers query efficiency.
- Repeatedly stored in each time partition
- Repeatedly accessed across multiple time partitions from cloud storage

Outline

- Background
- Motivation
- CloudTS Architecture
- Experiments
- Conclusion

CloudTS Architecture

- A lightweight timeseries storage model
- Seamless integration with existing TSDBs

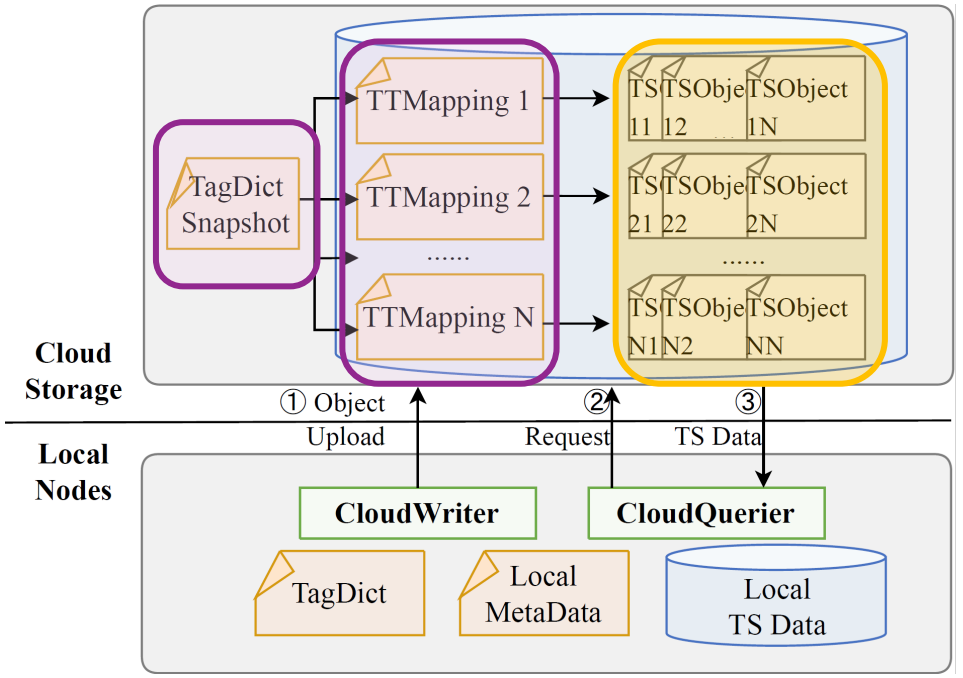
- Separate management of metadata and data

Metadata

- *TagDict*: A global tag dictionary for efficient tag management.
- *TTMapping*: A two-dimensional bitmap for timeseries-tag indexing.

Data

- *TSObject*: A chunked and ordered data storage unit for efficient cloud access.

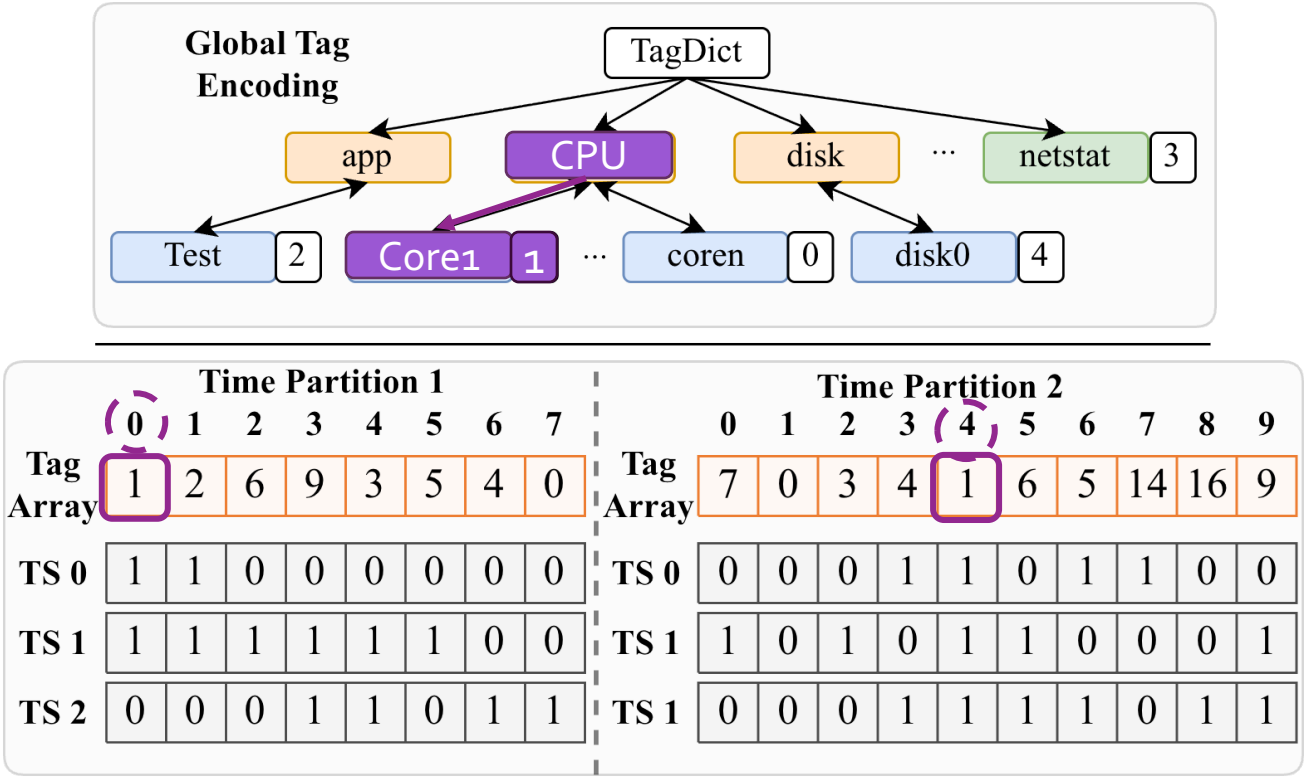


CloudTS-based timeseries system design.

TagDict: Global Tag Management

- Patricia Trie-like structure for tag management.
- Global encoding to minimize redundancy.
- Time-partition-aware tag arrays for optimized query performance.

- For example, tag pair "*cpu = core1*" is resolved as global encoding "1".
- In TP1, the encoding is "0"; in TP2, the encoding is "4".

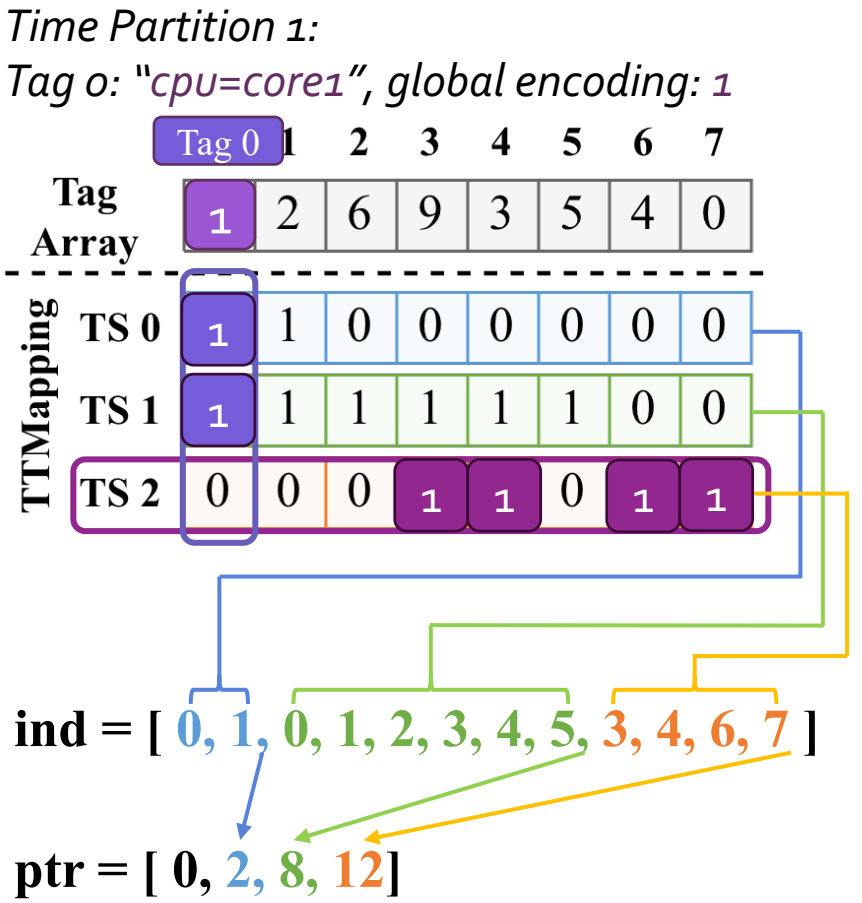


Tag management in local memory and cloud objects.

TTMapping: Timeseries-Tag Index

- One TTMapping per Time Partition.
- 2D bitmap to represent compact inverted index.
- Bidirectional indexing for timeseries <-> tag lookups.

- High space Overhead
  - Sparse matrix -> Compress sparse row (CSR)
  - Binary values ('0' and '1') -> Exclude "value" array
  - Timeseries Metadata Mapping Compression (TMMC):
    - $ind = \{j | B_{ij} = 1, 0 \leq i \leq M, 0 \leq j \leq N\}$
    - $ptr_k = \begin{cases} 0, & if\ k = 0 \\ \sum_{i=0}^{i=k} \sum_{j=0}^{j=N} B_{ij,} & if\ 1 \leq k \leq M \end{cases}$
    - M – timeseries, N – tags, B – Bitmap



TSObject: Data Organization

- Performance monitoring timeseries queries typically access a small subset of correlated timeseries over aligned time ranges
  - E.g., "GET /api/v1/query_range?query={ts_metrics ="cpu_usage",node="node01"}& start=2025-07-01T20:10:30.781Z&end=2025-07-01T20:11:00.781Z"
- Trade-off between object management and access speed
  - Fewer TS each object -> More TSObject lookup overhead
  - More TS each object-> High read amplification
- Group-based TSObjects (details in the paper)
  - Key idea: Group related timeseries into one TSObject based on tag similarity and tag selectivity
  - Maximize query locality

Outline

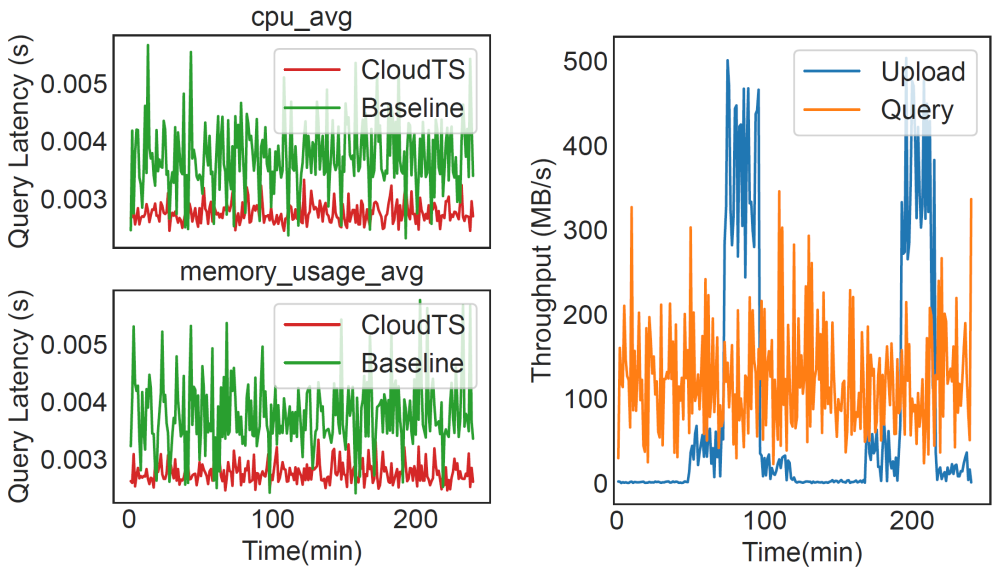
- Background
- Motivation
- CloudTS Architecture
- Experiments
- Conclusion

Evaluation Setup

- Integrated CloudTS into Cortex
- Environment
  - Amazon cloud environment
  - An Amazon EC2 Ubuntu 22.04 server with 2.5GHz Intel Xeon Platinum 8259CL CPU, 32 GB RAM
  - Amazon S3 storage
- Comparisons
  - Cortex
  - Parquet
  - JSON Time Series (JTS)

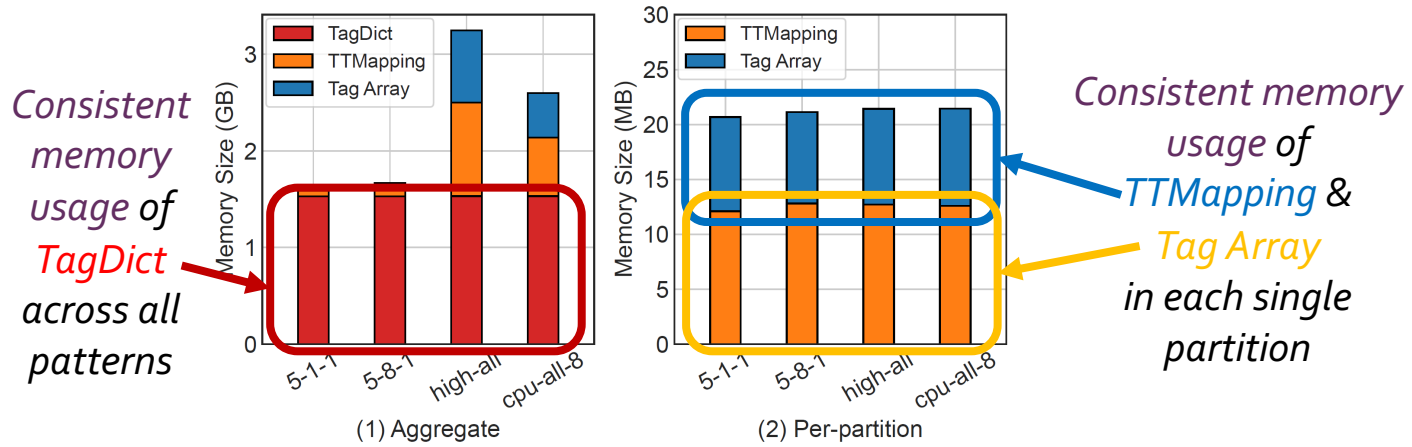
Production Environment Evaluation

- End-to-End Query Latency:
  - 1.43× reduced end-to-end query latency
- Data Transferring Throughput:
  - Stable query throughput (~130 MB/s)
  - Periodically uploads data for every time partition (two hours).

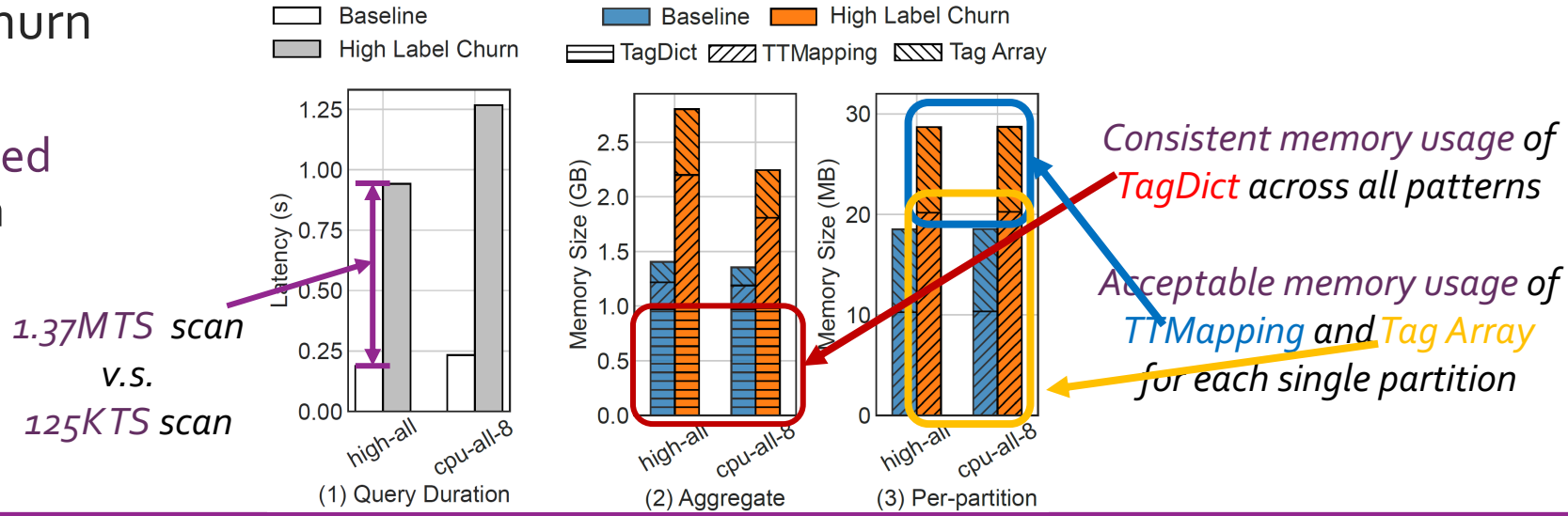


Scalability Assessment

- Long-term retention condition
  - Spanning 200 partitions (200 × 2=400 hours)
  - Total 1.36 million unique timeseries



- High-frequency label churn
  - 200 partitions
  - Up to 3 million short-lived timeseries per partition



Outline

- Background
- Motivation
- CloudTS Architecture
- Experiments
- Conclusion

Conclusion

- CloudTS
  - A timeseries storage model designed for cloud storage
- Key Idea
  - Metadata - data separation

    -> Reduce read amplification
  - Novel timeseries metadata management: TagDict and TTMapping

    -> Reduce metadata redundancy
- Benefits
  - Up to 36% lower query latency and 60% less data access

Thanks!

Q&A