Peregreen – modular database for efficient storage of historical time series in cloud environments

Alexander A. Visheratin, Alexey Struckov, Semen Yufa,
Alexey Muratov, Denis Nasonov, Nikolay Butakov
ITMO University

Yury Kuznetsov, Michael May
Siemens

Motivation

Use cases:

- **Browsing** large amounts of historical data – quick zoom-in and zoom-out in long time intervals.
- Conditional **search** for the data (e.g. anomalies search) – complex conditions, visualization of results.

Requirements

- Store terabytes of time series data for tens of years.
- 1 second for conditional search in whole database.
- Execution speed of 450,000 entries per second per node.
- All data must be stored in Amazon S3.

Proposed approach

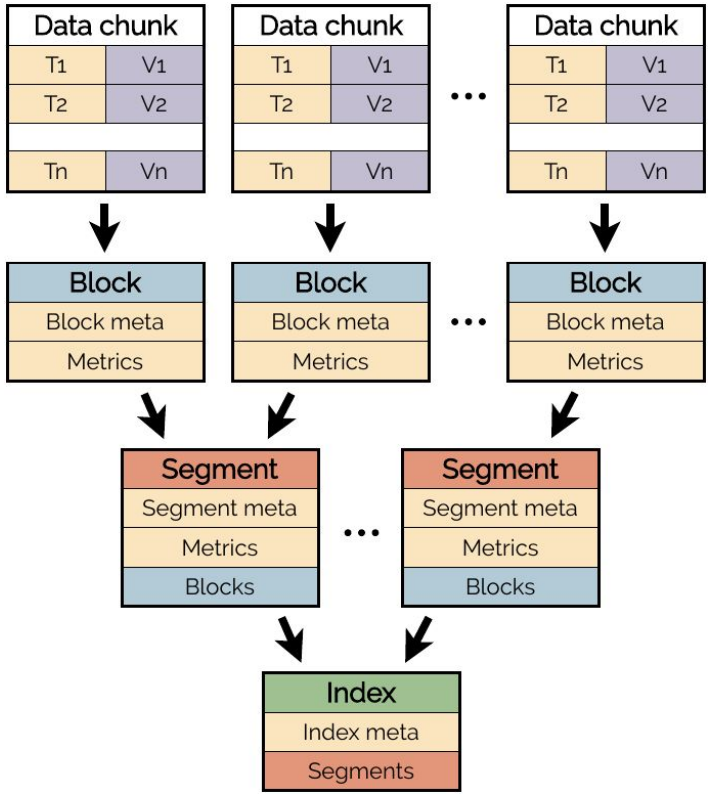
Main goal: minimize network overheads for search and extraction.

Challenges:

- Precise navigation in files at the backend storage.
- Minimize the number of requests for reading and writing.
- Minimize the size of index and data files.

Three-tier data indexing

- Split data into chunks by equal time steps (block interval).
- Create index blocks with meta for extraction and metrics for search.
- Combine blocks into segments by equal time steps (segment interval).
- Combine segments into the sensor index.



Index metadata

Block:

- Number of elements (preallocate arrays during extraction).
- Length of the encoded data (navigate and read data from backend storage).
- Compression flag (whether to decompress binary data).

Segment:

- Unique identifier (file name in backend storage).
- Start time (search and extraction).
- Version (storage consistency).

Index:

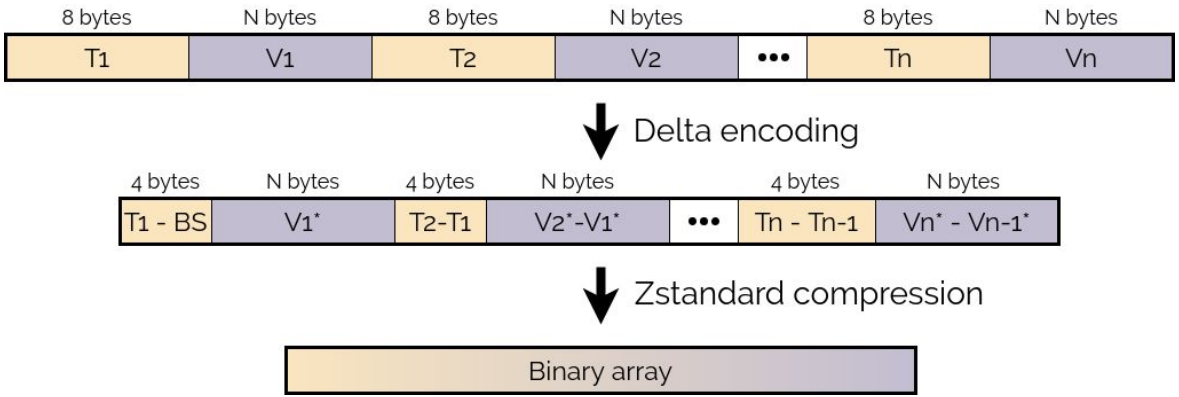
- Unique identifier (sensor name).
- Data type (int32, float32, int64, etc.)
- Block and segment intervals (CRUD operations).
- Version (storage consistency).

Read-optimized series encoding (I)

Two-step process:

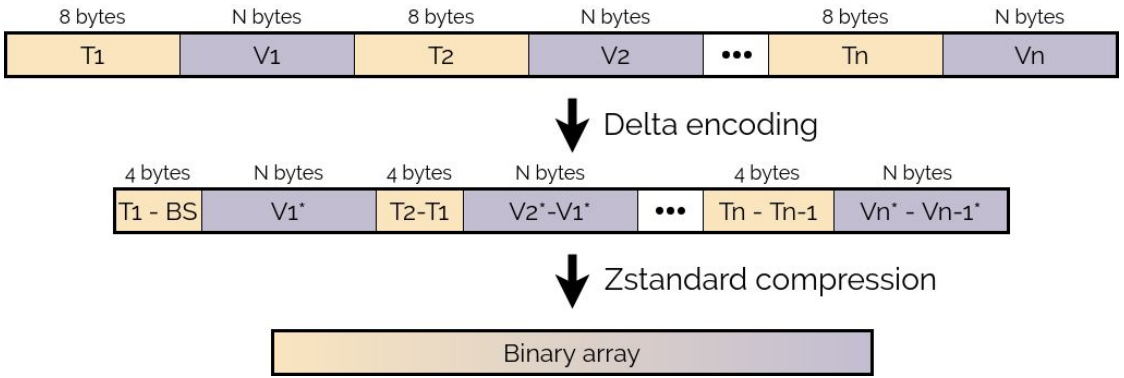
1. Delta encoding for timestamps and values.
2. (Optional) Byte array compression using Zstandard algorithm.

Compressed byte arrays are stored in segment files.

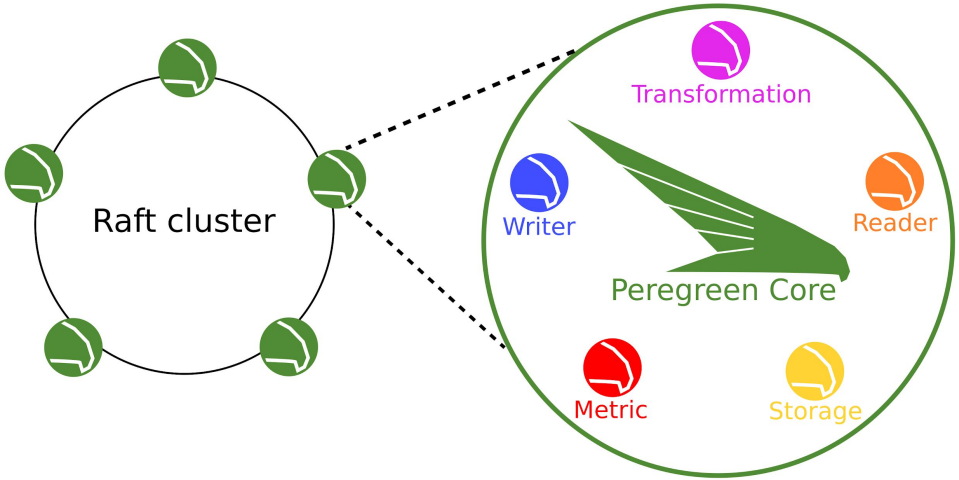


Read-optimized series encoding (II)

- Timestamp-value pairs are stored together (extraction with one read).
- Timestamp deltas are stored as unsigned int32 (maximum 49 days in ms).
- For values delta - IEEE 754 binary representations (longer zero-filled bit sequences).
- Zstandard algorithm gives 2x compression and 10-15% overhead.



Peregreen overview



- Core implements indexing and encoding logic.
- Upload, delete, search and extract operations are supported via HTTP API.
- Modules as programming interfaces for easy adding new functionality.
- Raft cluster with replication to achieve consistency and fault-tolerance.

Peregreen core. Upload

1. **Split** input data into chunks. Load existing data if needed.
2. **Generate** index and data blocks.
3. **Combine** data blocks to segment files.
4. **Combine** index blocks to segments and write to index.
5. **Update** versions of segments and index.
6. **Write** segment files and index to backend storage.

Peregreen core. Search

- Performed using only index.
- Time granularity is the block interval.
- Core checks time and value conditions in segments and blocks.
- Time condition - interval set by user in the request.
- Value condition - queries consisting of metric conditions combined by logic operations.

Query example:
min lte 200 & max gte 50 | avg eq 75
(1) minimum metric is lower or equal to 200 and maximum metric is greater or equal to 50, or
(2) average metric is equal to 75.

Peregreen core. Extraction (I)

Four types of extraction:

1. **Full** - all values within an interval.
2. **Sampling** - points, timestamps of which differ by user-specified interval.
3. **Aggregation** - calculate a set of metrics for every user-specified interval.
4. **Transformation** - apply some functions to the extracted data.

Peregreen core. Extraction (II)

1. Calculate segments to use based on start and finish timestamps. Uses segment start times and segment interval.
2. Select blocks to be extracted. Uses segment start time and block interval.
3. Calculate range of bytes to be loaded from segment files in backend storage. Uses encoded data lengths from blocks.
4. Extract the data.
5. (Optional) Decompress the data.
6. Read the data and apply required functions during reading.

Peregreen modules

Integration with backend storage - indices reading and writing, writing segment files and reading parts of segment files.

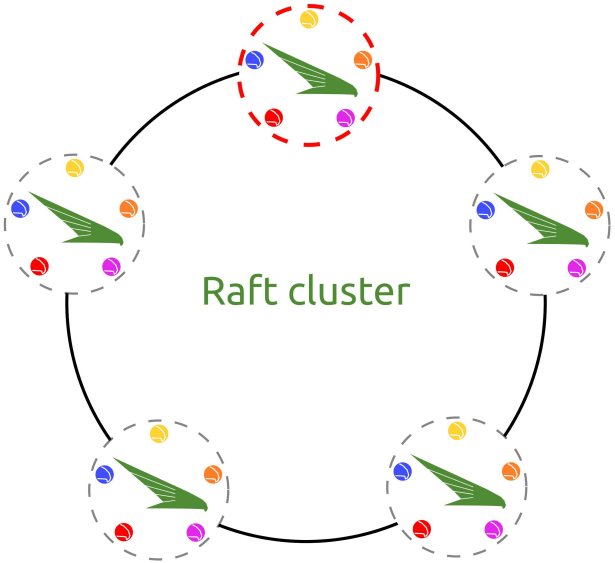
Reading data elements from the input stream one by one. Create data chunks on the fly.

Converting results - search results, timestamp-value pairs, aggregations or transformations - into desirable output format.

Aggregated statistical characteristics for the data. Can be combined (segment and block metrics).

Change the data as it is extracted.

Peregreen cluster



- High horizontal scalability.
- Sensors are distributed equally across nodes.
- Sensors replication for fault tolerance.
- Write requests go through leader node that applies lock on the sensor.
- Read requests are redirected to the responsible node.

Experiments

1. On-premise: Blade servers, Intel Xeon with 32 vCPUs, 128 GB RAM. 1, 4, 7 nodes. Compared to **ClickHouse and InfluxDB**.
2. Amazon EC2: c5.4xlarge instances with 16 vCPUs, 32 GB RAM and 500 GB attached EBS storages. 1, 4, 7 nodes. Compared **EBS vs S3** as backend storages.

On-premise: Uploading

Initial data volume - 750 GB.

Stored data volume, GB

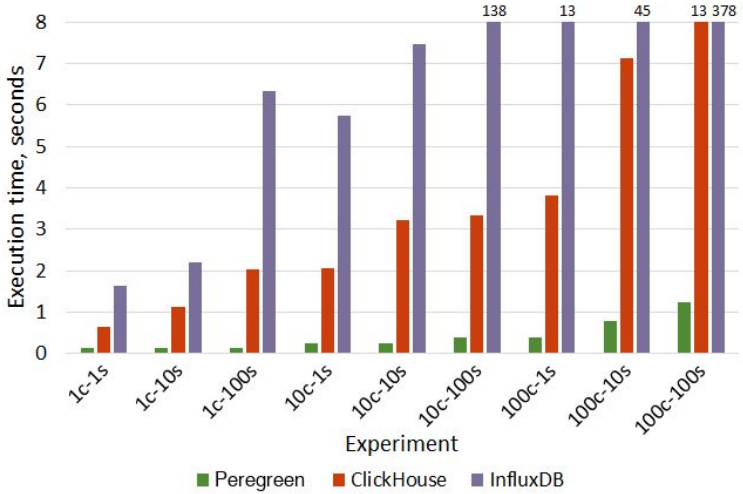
Peregreen	ClickHouse	InfluxDB
172	301	272

Uploading time, minutes

	1 instance	4 instances	7 instances
Peregreen	891	243	150
ClickHouse	**530**	**193**	**83**
InfluxDB	1847	583	N/A

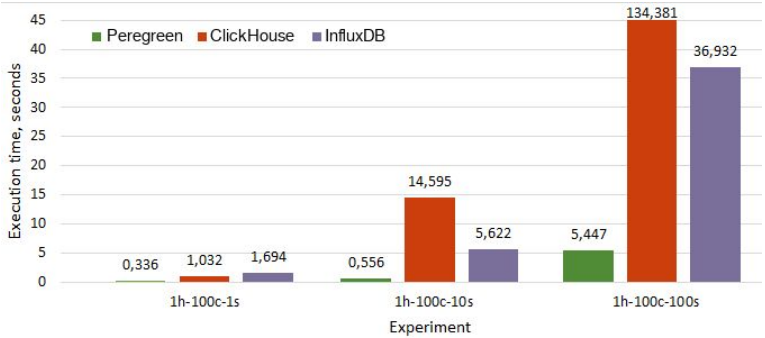
On-premise: Search and extraction

Data search

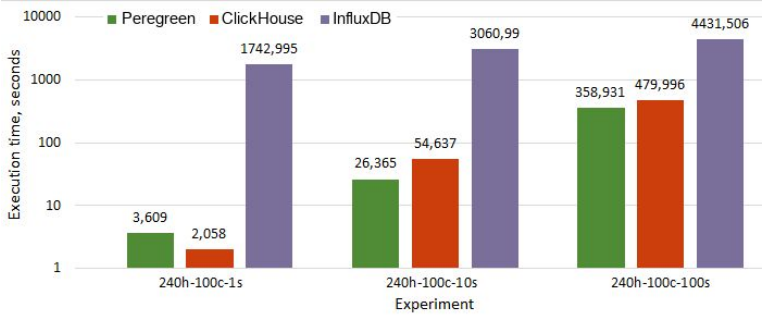


Data extraction

Small requests (1 hour)



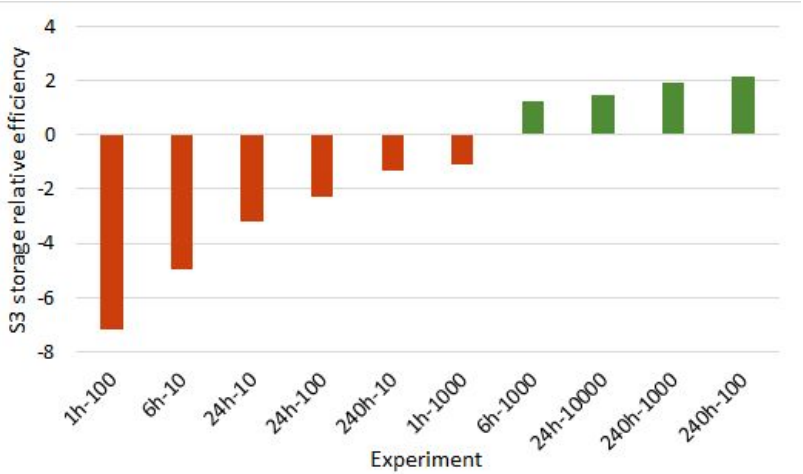
Large requests (10 days)



Y axis in log scale

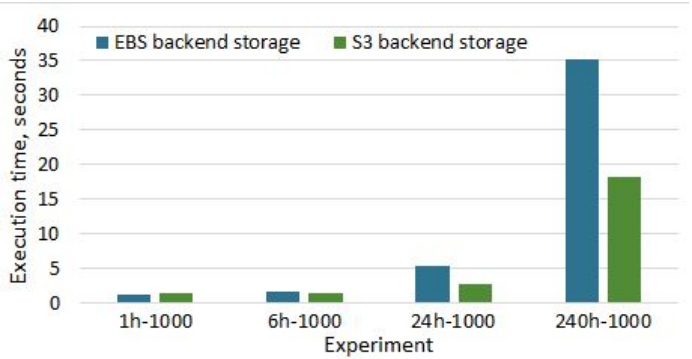
Amazon EC2: Extraction

S3 efficiency compared to EBS

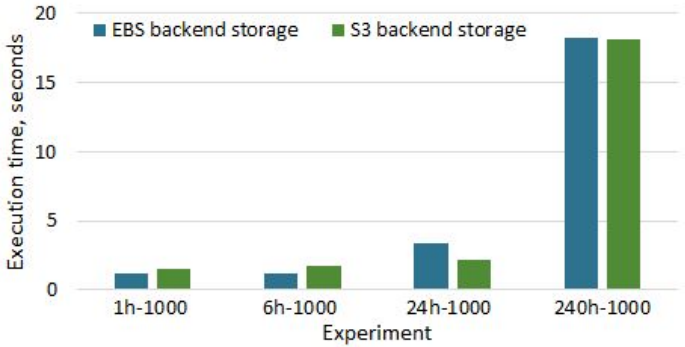


1000 concurrent requests for 1, 6, 24 and 240 hours.

4 instances:



7 instances:



Conclusion

Peregreen - fast time series database designed for efficient work with backend storage:

- Three-tier data indexing for small and powerful indices.
- Read-optimized series encoding for compact data representation.
- 5 types of modules for great extensibility.
- Fast uploading, search and extraction in both on-premise and cloud setups.

Thank you for your attention!

ITMO UNIVERSITY

Peregreen – modular database for efficient storage of historical time series in cloud environments

Alexander A. Visheratin
ITMO University
alexvish91@gmail.com

SIEMENS
Ingenuity for life

Acknowledgements

This work is carried out in the Siemens-ITMO Industrial Cognitive Technologies Lab and is funded by Siemens.