
How ClickHouse Inspired Us to Build a High Performance Time Series Database
OSA CON
2021

About me

- I’m Aliaksandr Valialkin, CTO and core developer at VictoriaMetrics
- I like programming in Go
- I’m fond of performance optimizations
- I like ClickHouse :)
- Follow me @GitHub - https://github.com/valyala

What is VictoriaMetrics?

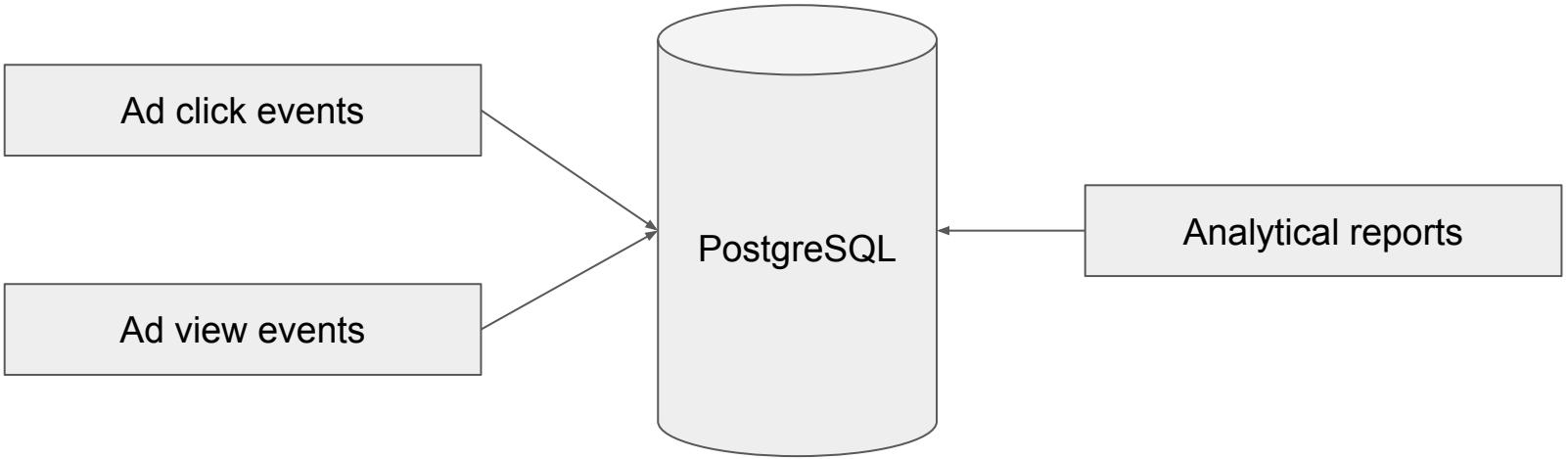
- Easy to use time series database and monitoring solution
- Can be used as drop-in replacement for Prometheus and Graphite
- Provides monitoring-optimized query language - MetricsQL (inspired by PromQL)
- Optimized for low resource usage (disk space, disk IO, CPU, RAM)
- Optimized for high performance

VICTORIA METRICS

The history of VictoriaMetrics

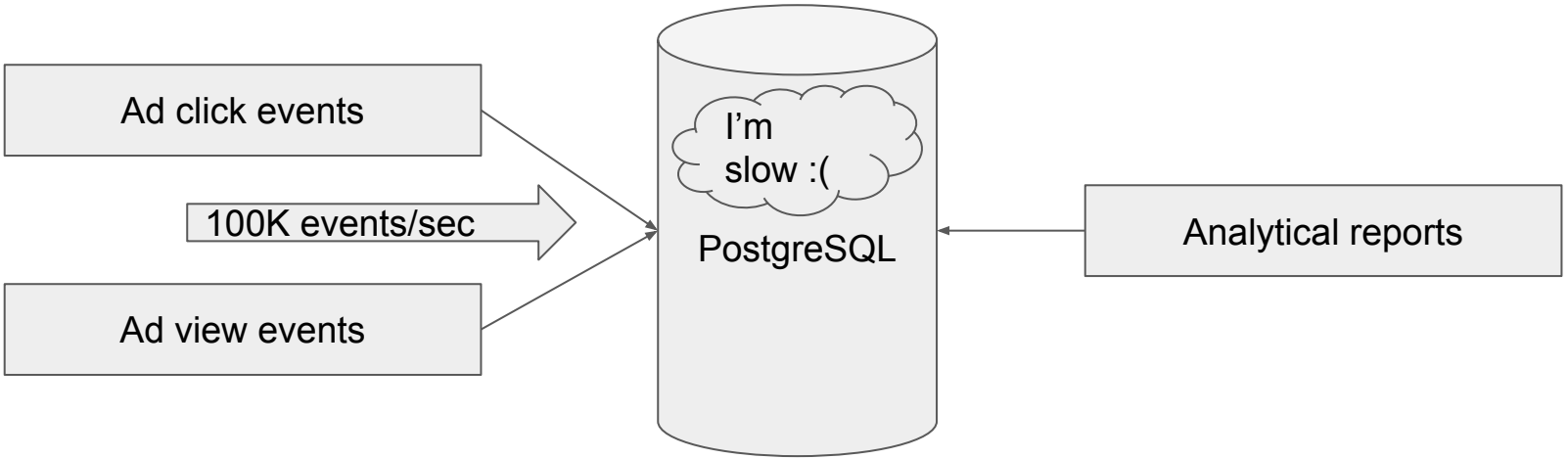
The history of VictoriaMetrics: ad analytics and PostgreSQL

- Long time ago we were using PostgreSQL for ad analytics



The history of VictoriaMetrics: ad analytics and PostgreSQL

- Long time ago we were using PostgreSQL for ad analytics
- PostgreSQL was great until the stream of the stored events start exceeding 100K rows/sec



The history of VictoriaMetrics: ad analytics and PostgreSQL

- We squeezed the maximum possible data ingestion speed from PostgreSQL: https://gist.github.com/valyala/ae3cbfa4104f1a022a2af9b8656b1131

Optimizing postgresql table for more than 100K inserts per second

- Create `UNLOGGED` table. This reduces the amount of data written to persistent storage by up to 2x.
- Set `WITH (autovacuum_enabled=false)` on the table. This saves CPU time and IO bandwidth on useless vacuuming of the table (since we never `DELETE` or `UPDATE` the table).
- Insert rows with `COPY FROM STDIN`. This is the fastest possible approach to insert rows into table.
- Minimize the number of indexes in the table, since they slow down inserts. Usually an index on `time timestamp with time zone` is enough.
- Add `synchronous_commit = off` to `postgresql.conf`.
- Use table inheritance for fast removal of old data:

The history of VictoriaMetrics: ad analytics and PostgreSQL

- There were issues with query performance as well
- These issues were solved by creating many tables with aggregate analytics
- The number of different aggregate tables was constantly growing...
- ...as well as the time needed for aggregate calculations

The history of VictoriaMetrics: ad analytics and PostgreSQL

- We started searching for a new database for ad analytics
- CitusDB looked promising. But it was hard to setup and operate
- Postgres-XL was even harder to setup and operate
- We tried MemSQL. It failed because of high RAM usage :(

The history of VictoriaMetrics: ad analytics and PostgreSQL

- We started searching for a new database for ad analytics
- CitusDB looked promising. But it was hard to setup and operate
- Postgres-XL was even harder to setup and operate
- We tried MemSQL. It failed because of high RAM usage :(
- Then we discovered ClickHouse!

ClickHouse for ad analytics

The history of VictoriaMetrics: ad analytics and ClickHouse

- ClickHouse fit perfectly for our use case!
- We successfully migrated ad analytics from PostgreSQL to ClickHouse in one month
- ClickHouse was accepting up to 300K events per second on a single server with 50 columns per event
- It easily scaled to a cluster of a dozen of nodes, which was accepting 3M events/sec without issues
- ClickHouse could scan up to 1000x more rows per second than PostgreSQL on the same hardware!
- We dropped the majority of the pre-computed aggregate tables, since ClickHouse could calculate arbitrary reports for our ad analytics over raw events in a blink of an eye!

The history of VictoriaMetrics: ad analytics and ClickHouse

- We were happy users of ClickHouse, so we decided to give back to community

The history of VictoriaMetrics: ad analytics and ClickHouse

- We were happy users of ClickHouse, so we decided to give back to community
- We created a datasource for Grafana, which allowed building graphs and reports in Grafana from ClickHouse data. Now it is supported by Altinity - https://github.com/Vertamedia/clickhouse-grafana

The history of VictoriaMetrics: ad analytics and ClickHouse

- We were happy users of ClickHouse, so we decided to give back to community
- We created a datasource for Grafana, which allowed building graphs and reports in Grafana from ClickHouse data. Now it is supported by Altinity - https://github.com/Vertamedia/clickhouse-grafana
- Then we created a proxy, which could be used for authorizing and load-balancing of insert and select requests among ClickHouse clusters, replicas and nodes - https://github.com/Vertamedia/chproxy

The history of VictoriaMetrics: ad analytics and ClickHouse

- We were happy users of ClickHouse, so we decided to give back to community
- We created a datasource for Grafana, which allowed building graphs and reports in Grafana from ClickHouse data. Now it is supported by Altinity - https://github.com/Vertamedia/clickhouse-grafana
- Then we created a proxy, which could be used for authorizing and load-balancing of insert and select requests among ClickHouse clusters, replicas and nodes - https://github.com/Vertamedia/chproxy
- Then I created a fast reader of TSV data from ClickHouse - https://github.com/valyala/tsvreader

The history of VictoriaMetrics: ad analytics and ClickHouse

- We were happy users of ClickHouse, so we decided to give back to community
- We created a datasource for Grafana, which allowed building graphs and reports in Grafana from ClickHouse data. Now it is supported by Altinity - https://github.com/Vertamedia/clickhouse-grafana
- Then we created a proxy, which could be used for authorizing and load-balancing of insert and select requests among ClickHouse clusters, replicas and nodes - https://github.com/Vertamedia/chproxy
- Then I created a fast reader of TSV data from ClickHouse - https://github.com/valyala/tsvreader
- I filed a few feature requests and bug reports at ClickHouse repository - https://github.com/ClickHouse/ClickHouse/issues?q=is%3Aissue+author%3Avalyala

1 Open ✓ 13 Closed — Author — Label — Projects — Milestones — Assignee — Sort

- **feature: automatically convert string column block with low cardinality into int column block with `id -> value` mapping before compression** feature performance — 14
  #1567 by valyala was closed on Jan 24, 2019
- **feature: don't waste page cache resources when merging big parts** enhancement — 4
  #1566 by valyala was closed on Oct 31, 2018
- **Reduce memory usage during vertical merge** — 6
  #931 by valyala was closed on Aug 21, 2017
- **Feature request: add ability to apply delta or delta-of-delta encoding to numeric columns before compression** feature — 18
  #838 by valyala was closed on Jan 27, 2019
- **Distributed table engine cannot merge distinctly ordered columns** — 5
  #829 by valyala was closed on Oct 17, 2018
- **Cannot fetch dictionary by https from a server with valid ECDSA certificate** comp-dictionary — 2
  #820 by valyala was closed on Oct 25, 2017
- **Use MergeTree key when locating rows on `key_column IN (subquery)` conditions** enhancement — 4
  #687 by valyala was closed on Oct 17, 2018
- **Feature request: evaluate and prune constant expressions before columns' scanning** comp-optimizers performance — 5
  #658 opened on Apr 6, 2017 by valyala
- **Feature request: add ability to drop MergeTree table parts by primary key range** comp-mutations — 3
  #654 by valyala was closed on Jul 16, 2018
- **Slow batch inserts into Buffer table** enhancement — 2
  #594 by valyala was closed on Mar 23, 2017
- **FR: arbitrary min-max indices** feature — 2
  #530 by valyala was closed on Jul 18, 2019
- **Error when filtering on updated enum column type** bug — 1
  #512 by valyala was closed on Mar 9, 2017
- **ALTER TABLE ... FREEZE PARTITION sometimes returns 'File not found' error** — 3
  #393 by valyala was closed on Oct 12, 2017
- **Unable to add new values to Enum column if it belongs to primary key** enhancement — 3
  #364 by valyala was closed on Feb 17, 2017

The history of VictoriaMetrics: Prometheus and ClickHouse

- We were using Zabbix for infrastructure and application monitoring
- The experience wasn't very good, so we were searching for a replacement
- We discovered Prometheus in the beginning of 2017
- It was great!
- Unfortunately it started to slow down with the increased number of monitored apps
- So we decided to try ClickHouse as a remote storage for Prometheus

ClickHouse as data storage for Prometheus

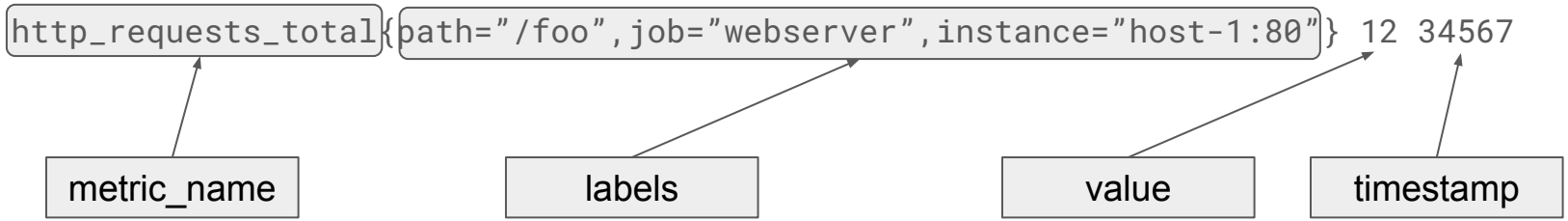
The history of VictoriaMetrics: ClickHouse as metrics storage

- Prometheus collects samples with the following structure:

```
http_requests_total{path="/foo",job="webserver",instance="host-1:80"} 12 34567
```

The history of VictoriaMetrics: ClickHouse as metrics storage

- Prometheus collects samples with the following structure:



The history of VictoriaMetrics: ClickHouse as metrics storage

- Let's store Prometheus samples in the following ClickHouse table:

```
samples (
  metric_name String,
  labels String,
  timestamp Int64,
  value Float64
) ORDER BY (metric_name, labels, timestamp)
```

The history of VictoriaMetrics: ClickHouse as metrics storage

- Let's store Prometheus samples in the following ClickHouse table:

```
samples (
  metric_name String,
  labels String,
  timestamp Int64,
  value Float64
) ORDER BY (metric_name, labels, timestamp)
```

- This allows quickly locating samples for a particular metric
- This allows searching for samples with particular labels via regexp matching. But this isn't an easy and fast task
- `metric_name` and `labels` are duplicated with every sample

The history of VictoriaMetrics: ClickHouse as metrics storage

- Let's move `metric_name` and `labels` columns into a separate table:

```
metric_to_series (metric_name String, labels String, series_id Uint64)
ORDER BY (metric_name, labels)
```

The history of VictoriaMetrics: ClickHouse as metrics storage

- Let's move `metric_name` and `labels` columns into a separate table:

```
metric_to_series (metric_name String, labels String, series_id Uint64)
ORDER BY (metric_name, labels)
```

- Then the `samples` table will look like:

```
samples (series_id Uint64, timestamp Int64, value Float64) ORDER BY
(series_id, timestamp)
```

- This reduces the overhead for storing `metric_name` and `labels` per each sample

The history of VictoriaMetrics: ClickHouse as metrics storage

- Let's move `metric_name` and `labels` columns into a separate table:

```
metric_to_series (metric_name String, labels String, series_id Uint64)
ORDER BY (metric_name, labels)
```

- Then the `samples` table will look like:

```
samples (series_id Uint64, timestamp Int64, value Float64) ORDER BY
(series_id, timestamp)
```

- This reduces the overhead for storing `metric_name` and `labels` per each sample
- But it is still hard to search by labels

The history of VictoriaMetrics: ClickHouse as metrics storage

- Let’s introduce an additional table:

```
label_to_series (label_name_value String, series_id Uint64) ORDER BY
(label_name_value, series_id)
```

- The `label_name_value` column contains `label=value` strings
- This table simplifies and speeds up searching by labels
- Such table is known as “inverted index”

The history of VictoriaMetrics: ClickHouse as metrics storage

- The `samples` table can be optimized further with special codecs for timestamp and value:

```
samples (timestamp Int64 Codec(DoubleDelta), value Float64 Codec(Gorilla))
```

- These codecs reduce disk space usage for typical time series data

The history of VictoriaMetrics: ClickHouse as metrics storage

- The `samples` table can be optimized further with special codecs for timestamp and value:

```
samples (timestamp Int64 Codec(DoubleDelta), value Float64 Codec(Gorilla))
```

- These codecs reduce disk space usage for typical time series data
- Unfortunately these codecs weren't available in ClickHouse in 2017 :(
- These codecs were added after the feature request - https://github.com/ClickHouse/ClickHouse/issues/838

The history of VictoriaMetrics: ClickHouse as metrics storage

- The `series` and `label_to_series` tables also can be optimized further by using LowCardinality codecs:

```
series (metric_name LowCardinality(String), labels String, series_id Uint64)

label_to_series (label_name_value LowCardinality(String), series_id Uint64)
```

- This saves disk space and improves query performance

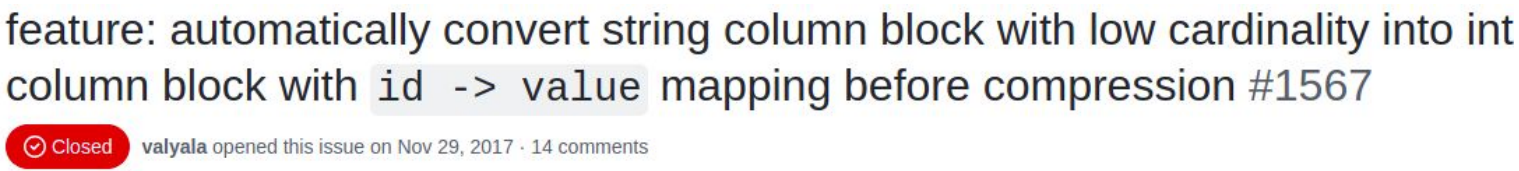
The history of VictoriaMetrics: ClickHouse as metrics storage

- The `series` and `label_to_series` tables also can be optimized further by using LowCardinality codecs:

```
series (metric_name LowCardinality(String), labels String, series_id Uint64)

label_to_series (label_name_value LowCardinality(String), series_id Uint64)
```

- This saves disk space and improves query performance
- But LowCardinality codec wasn't available in ClickHouse in 2017 :(
- It was added as a response to the feature request - https://github.com/ClickHouse/ClickHouse/issues/1567



The history of VictoriaMetrics: ClickHouse as metrics storage

- Further optimizations:
  - The `metric_name` column can be encoded as a label with a special name - `__name__` (Prometheus does this)
  - There should be a table for fast lookup of metric_name{labels} from `series_id` during queries

The history of VictoriaMetrics: ClickHouse as metrics storage

- The final database structure would consist of the following tables:

```
metric_to_series (metric_name_with_labels String, series_id Uint64) ORDER
BY (metric_name_with_labels, labels)
```

This table is used by the app, which accepts new samples and needs to determine the corresponding series_id per each incoming sample

The history of VictoriaMetrics: ClickHouse as metrics storage

- The final database structure would consist of the following tables:

```
metric_to_series (metric_name_with_labels String, series_id Uint64) ORDER BY (metric_name_with_labels, labels)
```

```
series_to_metric (series_id Uint64, metric_name_with_labels String) ORDER BY (series_id)
```

This table is needed for converting the found series_id to human-readable format metric_name{labels} during queries

The history of VictoriaMetrics: ClickHouse as metrics storage

- The final database structure would consist of the following tables:

```
metric_to_series (metric_name_with_labels String, series_id Uint64) ORDER BY (metric_name_with_labels, labels)

series_to_metric (series_id Uint64, metric_name_with_labels String) ORDER BY (series_id)

label_to_series (label_name_value LowCardinality(String), series_id Uint64) ORDER BY (label_name_value, series_id)
```

This table is used for fast lookups of series_id for the given label filters during queries. For example, http_requests_total{job="webserver"} should find series_id values for series with both {__name__="http_requests_total"} and {job="webserver"} labels

The history of VictoriaMetrics: ClickHouse as metrics storage

- The final database structure would consist of the following tables:

```
metric_to_series (metric_name_with_labels String, series_id Uint64) ORDER BY (metric_name_with_labels, labels)

series_to_metric (series_id Uint64, metric_name_with_labels String) ORDER BY (series_id)

label_to_series (label_name_value LowCardinality(String), series_id Uint64) ORDER BY (label_name_value, series_id)

samples (series_id Uint64, timestamp Int64 Codec(DoubleDelta), value Float64 Codec(Gorilla)) ORDER BY (series_id, timestamp)
```

This table stores (timestamp, value) pairs for the ingested samples

The history of VictoriaMetrics: ClickHouse as metrics storage

- This approach looks good
- But it has the following issues:
  - It needs an external app for collecting the incoming samples and quickly adding missing entries to `metric_to_series`, `series_to_metric` and `label_to_series` tables. The app should be able to buffer incoming data in order to reduce the frequency of inserts to ClickHouse
  - It needs an external app for providing Prometheus-compatible querying API
  - This increases operational complexity
  - The on-disk compression level can be improved further

Meet VictoriaMetrics!

Meet VictoriaMetrics!

- So we decided to create a specialized time series database from scratch in order to solve the mentioned issues
- It should meet the following requirements:
  - It must be fast
  - It must be easy to setup and operate
  - It must scale both vertically (more CPU and RAM) and horizontally (multiple nodes)
  - It must be easy to code

Meet VictoriaMetrics!

- So we decided to create a specialized time series database from scratch in order to solve the mentioned issues
- It should meet the following requirements:
  - It must be fast
  - It must be easy to setup and operate
  - It must scale both vertically (more CPU and RAM) and horizontally (multiple nodes)
  - It must be easy to code
- I don’t like C++ because of its complexity, but I like writing fast code in Go
- So VictoriaMetrics is written in Go :)

Meet VictoriaMetrics!

- Conceptually VictoriaMetrics uses the same database scheme discussed previously:
  - metric_to_series (metric_name_with_labels string, series_id uint64)
  - series_to_metric (series_id uint64, metric_name_with_labels string)
  - label_to_series (label_name_value string, series_id uint64)
  - samples (series_id uint64, timestamp int64, value float64)
- But it doesn’t use external libraries or apps for the database
- It implements specially optimized persistent data structures for the given tables

VictoriaMetrics: persistent data structures

- The first three tables - `metric_to_series`, `series_to_metric` and `label_to_series` - are stored in a mergeset (aka indexdb). It stores sorted strings. It is optimized for fast data insertion, fast lookups and fast range scans by string prefix
- The last table - `samples` - is stored in a separate data structure similar to MergeTree in ClickHouse

VictoriaMetrics: data ingestion path

- VictoriaMetrics accepts samples in various formats: InfluxDB, Graphite, OpenTSDB, Prometheus, DataDog, CSV, JSON, etc.
- The ingested samples are buffered in memory
- The indexdb is dynamically updated when samples with new metric_name{labels} are ingested into the database
- The buffered samples are converted to ClickHouse-like parts and flushed to disk every second

VictoriaMetrics: `samples` table internals

- A table is split into per-month partitions. This allows instant removal of the data outside the retention period
- Each partition consists of ClickHouse-like parts, which are merged in background into bigger parts
- Each part is split into blocks
- Each block contains samples only for a single time series. Samples are sorted by timestamp in each block
- Block size is limited to 8K samples, so it can fit CPU cache for the max processing speed
- Blocks are processed independently of each other, so they can be processed in parallel
- `timestamp` and `value` columns are encoded separately with the most efficient codecs in order to get the maximum compression rate - https://faun.pub/victoriametrics-achieving-better-compression-for-time-series-data-than-gorilla-317bc1f95932

VictoriaMetrics: query path

- Query is split into two steps:
  - Selecting time series matching the given label filters on the given time range
  - Processing samples for the selected time series according to the query

VictoriaMetrics: query path

- Query is split into two steps:
  - Selecting time series matching the given label filters on the given time range
  - Processing samples for the selected time series according to the query
- Matching time series are searched via `label_to_series` table. The table may contain billions of entries, so VictoriaMetrics uses various optimization tricks (composite index, per-day index, optimized bitset, unpacked data cache, search cache) in order to speed up the search

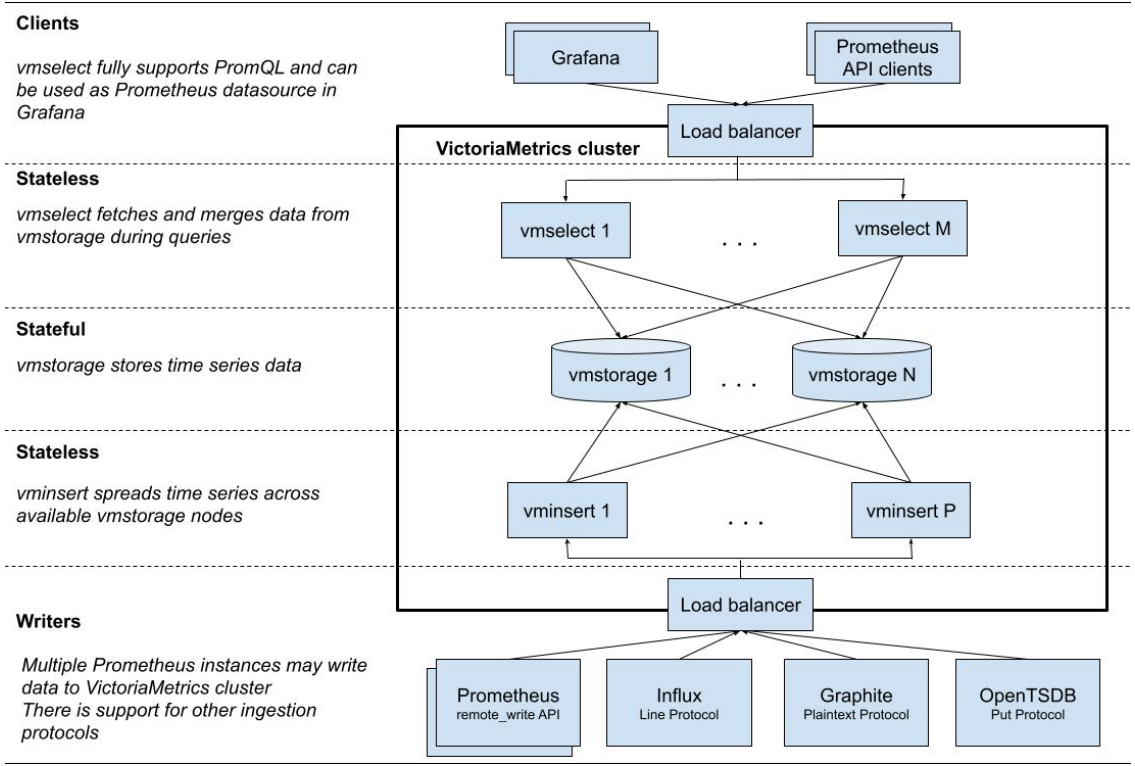
VictoriaMetrics: query path

- Query is split into two steps:
  - Selecting time series matching the given label filters on the given time range
  - Processing samples for the selected time series according to the query
- Matching time series are searched via `label_to_series` table. The table may contain billions of entries, so VictoriaMetrics uses various optimization tricks (composite index, per-day index, optimized bitset, unpacked data cache, search cache) in order to speed up the search
- Then VictoriaMetrics unpacks and processes blocks of samples for the found time series on the given time range. Blocks are unpacked and processed in parallel on all the available CPU cores in order to improve query performance

VictoriaMetrics: query path

- VictoriaMetrics implements MetricsQL - query language inspired by PromQL - https://docs.victoriametrics.com/MetricsQL.html
- Typical queries used in monitoring are easier to write in MetricsQL than in SQL - https://valyala.medium.com/promql-tutorial-for-beginners-9ab455142085
- VictoriaMetrics implements the following query APIs:
  - Prometheus API - https://docs.victoriametrics.com/#prometheus-querying-api-usage
  - Graphite API - https://docs.victoriametrics.com/#graphite-api-usage
- VictoriaMetrics can be used as drop-in replacement for Prometheus and Graphite

VictoriaMetrics: horizontal scalability



OSA CON 2021

VictoriaMetrics: the end result

- Fast time series database and monitoring solution inspired by ClickHouse
- Easy to setup and operate
- Integrates with Prometheus, Grafana, InfluxDB, Graphite, DataDog and OpenTSDB
- Scales vertically and horizontally
- Successfully handles tens of trillions (e.g. more than 10^13) of samples per node in production - https://docs.victoriametrics.com/CaseStudies.html

Thank you!