ClickHouse in Practice: Lessons from Building a Scalable Observability Backend

Nityananda Gohain & Srikanth Chekuri, Software Engineers @ SigNoz

About SigNoz

Schema Evolution in Logs and Traces

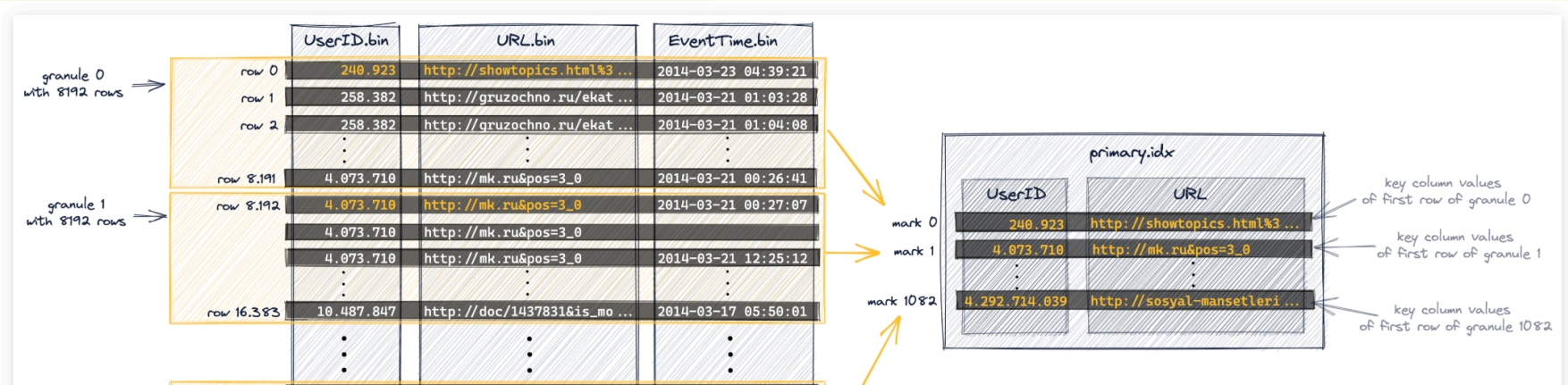
Indexes in ClickHouse

Two types of indexes in clickhouse

- Primary index (sparse index)
- Secondary index (skip index)

Primary Index (Sparse Index)

- Doesn't index every row, instead indexes on granules(blocks of rows)
- Allows to skip entire granules based on this index
- It is created based on the sorting key, which defines the order of storage.
- Eg of table with UserID, URL and EventTime column



Secondary Index (Skip Index)

- Helps improve query performance by reducing the number of granules scanned.
- Types are
  - Bloom filter
  - Set Index
  - MinMax Index
- Used for columns which are not present in primary key.

Logs Schema V1

```
CREATE TABLE signoz_logs.logs(
    `timestamp` UInt64 CODEC(DoubleDelta, LZ4),
    `id` String CODEC(ZSTD(1)),
    `body` String  CODEC(ZSTD(2)),
    `service_name` String CODEC(ZSTD(2)),
    . . . . . .
    . . . . . .
    INDEX idx service_name TYPE tokenbf_v1(10000, 3, 0) GRANULARITY 1
)
ENGINE = MergeTree
PARTITION BY toDate(timestamp / 1000000000)
ORDER BY (timestamp, id)
TTL toDateTime(timestamp / 1000000000) + toIntervalSecond(1296000)
```

Problems in schema

- Performance was not good enough in practice with skip index for the above schema.
- Projections based on most queried attributes.

```
PROJECTION default_projection
(
    SELECT *
    ORDER BY
        deployment_env,
        severity_text,
        service.name,
        component
)
```

Problems in schema

- Adding projections lead to increase in storage cost but it still didn't solve the problem in a scalable way.
- Dealing with projections was painful eg:- couldn't remove a column from projection, rename a column etc.
- ClickHouse may choose a projection which you may not want to use and that was difficult to manage.

Observations

- Users in most of the cases used a resource filter in their queries. Even if they don't resource filters can always be added to queries.
- Logs from different sources were lying in the same granule because of which the skip index on resource filters were not working.

Hierarchical Resource fingerprinting

- For all the logs of that pod the resource attributes will be same.
- We can define a hierarchical fingerprint though which we can store logs of same resource together.

Hierarchical Resource fingerprinting

Sample Logs

```
message = hello srikanth , cluster = c1, namespace = n1, pod = p1
message = bye srikanth   , ec2.tag.env = fn-prod, cloudwatch.log.stream = service1
message = hello ashu     , cluster = c1, namespace = n1, pod = p1
message = hello ankit    , cluster = c2, namespace = n1, pod = p1
message = hello nitya    , cluster = c1, namespace = n1, pod = p2
message = bye nitya      , ec2.tag.env = fn-prod, cloudwatch.log.stream = service1
message = hello vicky    , cluster = c1, namespace = n2, pod = p1
```

Hierarchical Fingerprints

```
cluster=c1;namespace=n1;pod=p1;hash=15482603570120227210
ec2.tag.env=fn-prod;cloudwatch.log.stream=service1;hash=10649409385811604510
cluster=c1;namespace=n1;pod=p1;hash=15382603570120226210
cluster=c2;namespace=n1;pod=p1;hash=15382603570120226210
cluster=c1;namespace=n1;pod=p1;hash=15182603570120224210
ec2.tag.env=fn-prod;cloudwatch.log.stream=service1;hash=10649409385811604510
cluster=c1;namespace=n1;pod=p1;hash=15282603570120223210
```

Hierarchical Resource fingerprinting

Final ordered Data

```
message = hello srikanth , cluster = c1, namespace = n1, pod = p1
message = hello ashu     , cluster = c1, namespace = n1, pod = p1
message = hello nitya    , cluster = c1, namespace = n1, pod = p2
message = hello vicky    , cluster = c1, namespace = n2, pod = p1
message = hello ankit    , cluster = c2, namespace = n1, pod = p1
message = bye nitya      , ec2.tag.env = fn-prod, cloudwatch.log.stream = service1
message = bye srikanth   , ec2.tag.env = fn-prod, cloudwatch.log.stream = service1
```

New Schema

- So eventually this is the primary key/sorting key that we came up with

```
ORDER BY (ts_bucket_start, resource_fingerprint, severity_text, timestamp, id)
```

Performance Difference

```
EXPLAIN indexes = 1
SELECT
    toStartOfInterval(fromUnixTimestamp64Nano(timestamp), toIntervalSecond(60)) AS ts,
    toFloat64(count(*)) AS value
FROM signoz_logs.logs
WHERE
    `resource_string_k8s$$namespace$$name` = 'dev'
    AND JSON_EXISTS(body, '$."event"."result"')
    AND (JSON_VALUE(body, '$."event"."result"') = 'complete')
GROUP BY ts
ORDER BY value DESC
SETTINGS max_threads = 8
```

- Old schema

```
┌─explain─────────────────────────────────────────────────┐
│ Expression (Projection)                                 │
│   Sorting (Sorting for ORDER BY)                        │
│     Expression (Before ORDER BY)                        │
│       MergingAggregated                                 │
│         Union                                           │
│           Aggregating                                   │
│             Expression (Before GROUP BY)                │
│               ReadFromMergeTree (signoz_logs.logs)      │
│               Indexes:                                  │
│                 MinMax                                  │
│                   Condition: true                       │
│                   Parts: 22/22                          │
│                   Granules: 41676/41676                 │
│                 Partition                               │
│                   Condition: true                       │
│                   Parts: 22/22                          │
│                   Granules: 41676/41676                 │
│                 PrimaryKey                              │
│                   Condition: true                       │
│                   Parts: 22/22                          │
│                   Granules: 41676/41676                 │
│                 Skip                                    │
│                   Name: resource_string_k8s$$namespace$$name_idx │
│                   Description: bloom_filter GRANULARITY 64       │
│                   Parts: 21/22                          │
│                   Granules: 41498/41676                 │
│           ReadFromRemote (Read from remote replica)     │
└─────────────────────────────────────────────────────────┘
```

- New Schema

```
┌─explain─────────────────────────────────────────────────────────────────────┐
│ CreatingSets (Create sets before main query execution)                      │
│   Expression (Projection)                                                   │
│     Sorting (Sorting for ORDER BY)                                          │
│       Expression (Before ORDER BY)                                          │
│         MergingAggregated                                                   │
│           Union                                                             │
│             CreatingSets (Create sets before main query execution)          │
│               Aggregating                                                   │
│                 Expression (Before GROUP BY)                                │
│                   Filter (WHERE)                                            │
│                     ReadFromMergeTree (signoz_logs.logs_v2) │               │
│                     Indexes:                                                │
│                       MinMax                                                │
│                         Condition: true                                     │
│                         Parts: 28/28                                        │
│                         Granules: 26135/26135                               │
│                       Partition                                             │
│                         Condition: true                                     │
│                         Parts: 28/28                                        │
│                         Granules: 26135/26135                               │
│                       PrimaryKey                                            │
│                         Keys:                                               │
│                           resource_fingerprint                              │
│                         Condition: (resource_fingerprint in 65-element set) │
│                         Parts: 28/28                                        │
│                         Granules: 222/26135                                 │
│             ReadFromRemote (Read from remote replica)                       │
└─────────────────────────────────────────────────────────────────────────────┘
```

Results

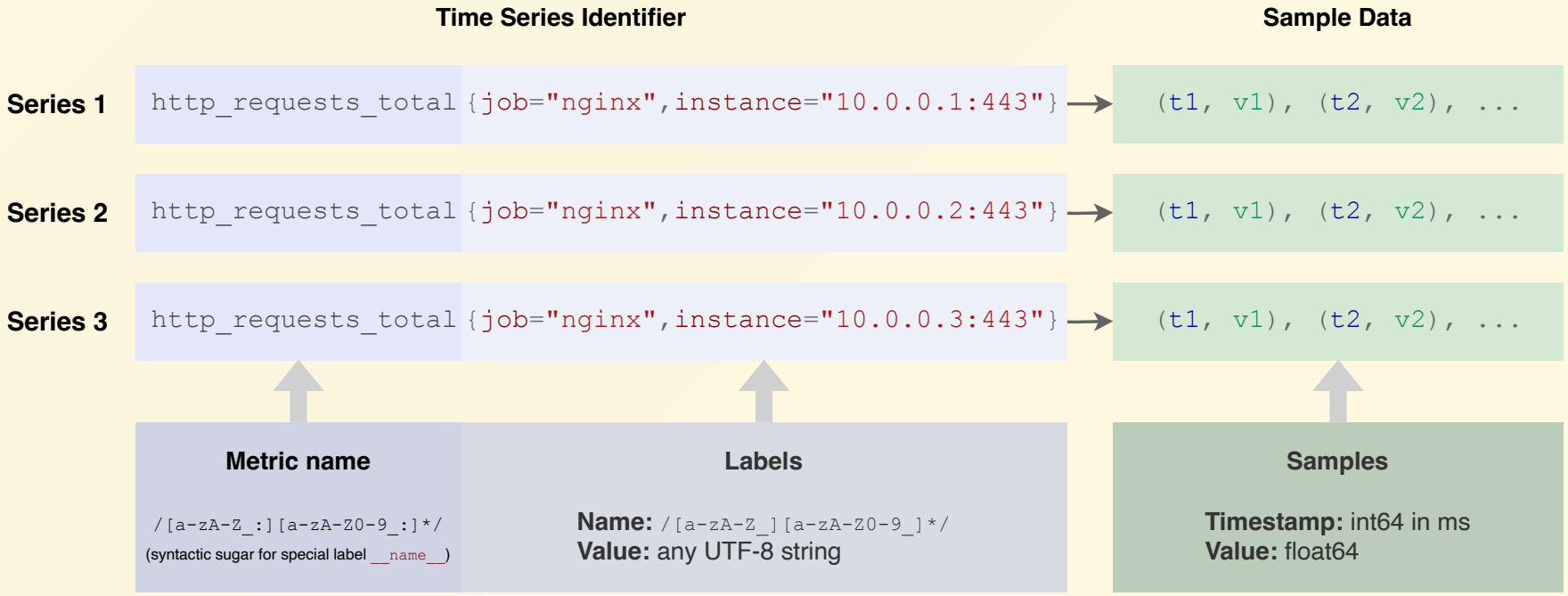
- Number of granules reduced by a magnitude for any query with resource attribute filtering.
- Reduced resource(CPU, disk) usage.

Metrics

- mainly used to see trends and patterns
- structured with consistent labels/dimensions
- highly compressed time-series data for storage efficiency
- support fast aggregation and summarization
- stored for a long time

Simplified data model

- **Sample**: a timestamp, a value
- **TimeSeries**: a name and a set of key-value pairs



Single table vs two tables

Single table

```
(
    `metric_name` LowCardinality(String),
    `fingerprint` UInt64 CODEC(DoubleDelta, LZ4),
    `timestamp_ms` Int64 CODEC(DoubleDelta, LZ4),
    `value` Float64 CODEC(Gorilla, LZ4),
    `labels` Map(LowCardinality(String), String) CODEC(ZSTD(2)),
)
ORDER BY (metric_name, fingerprint, timestamp_ms)
```

Issues with Single Table

- Maps access requires unpacking entire object
- Metric queries runs on billions of samples
- Extremely slow queries
- Storage is expensive

Possible improvements for single table

- Materialize and dematerialize

```
host_name string MATERIALIZED (labels['host_name'])
```

- Using ClickHouse Object JSON dataType.

Two tables

```
samples (
    `metric_name` LowCardinality(String),
    `fingerprint` UInt64 CODEC(Delta(8), ZSTD(1)),
    `unix_milli` Int64 CODEC(DoubleDelta, ZSTD(1)),
    `value` Float64 CODEC(Gorilla, ZSTD(1))
)
ORDER BY (metric_name, fingerprint, unix_milli)
```

```
timeseries (
    `metric_name` LowCardinality(String),
    `fingerprint` UInt64 CODEC(Delta(8), ZSTD(1)),
    `labels` String CODEC(ZSTD(5)),
)
ORDER BY (metric_name, fingerprint, unix_milli)
```

Materialized Views

```
SELECT
    metric_name,
    fingerprint,
    intDiv(unix_milli, 1800000) * 1800000 AS unix_milli,
    anyLast(last) AS last,
    min(min) AS min,
    max(max) AS max,
    sum(sum) AS sum,
    sum(count) AS count
FROM signoz_metrics.samples_v4_agg_5m
GROUP BY metric_name, fingerprint, unix_milli
```

Looking forward to

- Real merge join support
- New JSON data type

Thank you

We are looking to hire people across the teams.

Visit https://signoz.io/careers or shoot an email at hiring@signoz.io