TiDB Analyze

A Deep Dive

Based on TiDB v8.1.0

RUSTIN LIU

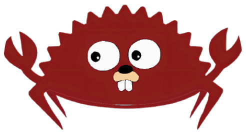
Press Space to Start

Rustin Liu

PingCAP Optimizer Team Member.

Cargo/Crates.io/Rustup Maintainer.

Tokio Console Maintainer.



- hi-rustin
- hi-rustin.rs

Agenda

Analyze Overview

Data Structure Overview

Data Flow Overview

Data Structure & Data Flow (TiKV Perspective)

Data Structure & Data Flow (TiDB Perspective)

Q&A

Analyze Overview

Analyze Statement

```
-- Analyze Tables
ANALYZE TABLE t1, t2;

-- Analyze Partitions
ANALYZE TABLE t PARTITION p1, p2;

-- Analyze Columns
ANALYZE TABLE t COLUMNS c1, c2;

-- Analyze Indexes
ANALYZE TABLE t INDEX idx1, idx2;

-- Analyze Partitions' Columns
ANALYZE TABLE t PARTITION p1 COLUMNS c1, c2;

-- Analyze Partitions' Indexes
ANALYZE TABLE t PARTITION p1 INDEX idx1, idx2;

-- Analyze Predicate Columns
ANALYZE TABLE t PREDICATE COLUMNS;

-- Analyze With Only 20 Top N
ANALYZE TABLE t COLUMNS c1, c2 WITH 20 TOPN;
```

Data Structure Overview

Data Structure Overview

A simple example.

Create table

```
use test;
create table t (a int);
```

Insert 2000 rows

```
import { Client } from "https://deno.land/x/mysql/mod.ts";

const client = await new Client().connect({...});

for (let i = 0; i < 2000; i++) {
  await client.execute(`INSERT INTO t (a) VALUES (?)`, [i]);
  if (i % 2 === 0) {
    await client.execute(`INSERT INTO t (a) VALUES (?)`, [i]);
  }
}

await client.close();
```

Data Structure Overview

Column Selectivity

```
explain select * from t where a = 100;
```

id	estRows	task	access object	operator info
TableReader_7	2.00	root		data:Selection_6
└─Selection_6	2.00	cop[tikv]		eq(test.t.a, 100)
└─TableFullScan_5	3000.00	cop[tikv]	table:t	keep order:false

```
func equalRowCountOnColumn(encodedVal []byte...) {
  rowcount, ok := c.TopN.QueryTopN(sctx, encodedVal)
  if ok {
    return float64(rowcount), nil
  }
}
```

Data Structure Overview

Column Selectivity

TopN

```
select * from mysql.stats_top_n order by value limit 5;
```

table_id	is_index	hist_id	value	count
106	0	1	0x038000000000000000	2
106	0	1	0x038000800000000002	2
106	0	1	0x038000800000000004	2
106	0	1	0x038000800000000006	2
106	0	1	0x038000800000000008	2

Data Structure Overview

Column Selectivity

```
explain select * from t where a = 1999;
```

id	estRows	task	access object	operator info
TableReader_7	1.00	root		data:Selection_6
└─Selection_6	1.00	cop[tikv]		eq(test.t.a, 1999)
└─TableFullScan_5	3000.00	cop[tikv]	table:t	keep order:false

```
func equalRowCountOnColumn(encodedVal []byte...) {
  histCnt, matched := c.Histogram.EqualRowCount(sctx, val, true)
  if matched {
    return histCnt, nil
  }
}
```

Data Structure Overview

Column Selectivity

```sql
select hist_id, bucket_id, count, repeats,
       CAST(lower_bound AS SIGNED) AS lower_bound,
       CAST(upper_bound AS SIGNED) AS upper_bound,
       ndv
from mysql.stats_buckets order by lower_bound desc limit 5;
```

hist_id	bucket_id	count	repeats	lower_bound	upper_bound	ndv
1	229	1	1	1999	1999	0
1	228	9	2	1993	1998	0
1	227	9	2	1987	1992	0
1	226	9	2	1981	1986	0
1	225	9	2	1975	1980	0

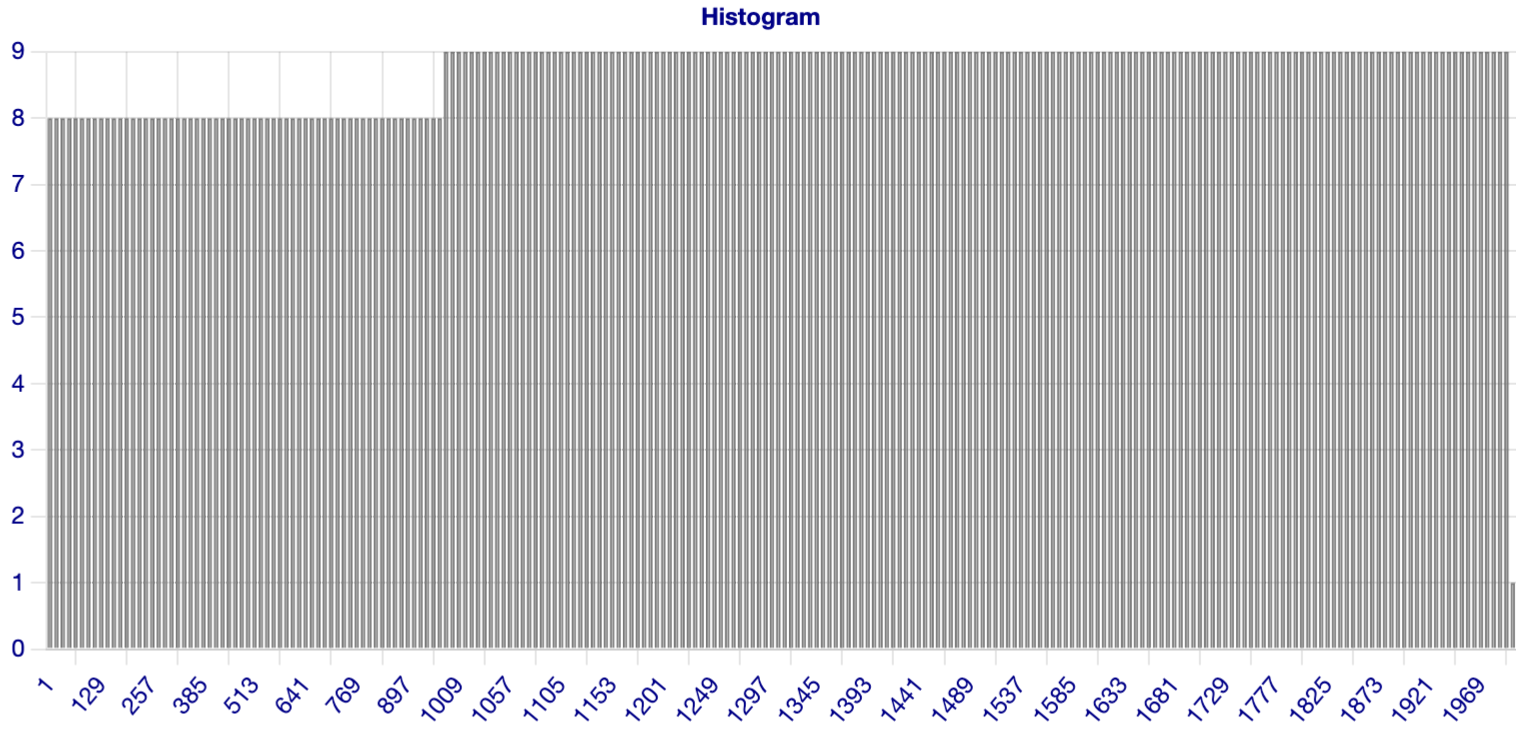
Data Structure Overview

Histogram Bucket

- Bucket ID: The bucket ID of the histogram.
- Count: The number of values till the bucket.(**cumulative**)
- Repeats: The number of repeated values at the upper bound.
- Lower Bound: The lower bound of the bucket.
- Upper Bound: The upper bound of the bucket.
- NDV: The number of distinct values in the bucket.(**Deprecated, always 0**)

```
{
    "bucket_id": 228,
    "count": 9,
    "repeats": 2,
    "lower_bound": 1993,
    "upper_bound": 1998,
    "ndv": 0
}
```

Data Structure [1]



[1] Piatetsky-Shapiro, Gregory, and Charles Connell. "Accurate Estimation Of The Number Of Tuples Satisfying A Condition"

Data Structure Overview

Column Selectivity

```
explain select * from t where a = 9999;
```

id	estRows	task	access object	operator info
TableReader_7	1.33	root		data:Selection_6
└─Selection_6	1.33	cop[tikv]		eq(test.t.a, 2000)
└─TableFullScan_5	3000.00	cop[tikv]	table:t	keep order:false

```
func equalRowCountOnColumn(encodedVal []byte...) {
  histNDV := float64(c.Histogram.NDV - int64(c.TopN.Num()))
  if histNDV <= 0 {
    return 0, nil
  }
  return c.Histogram.NotNullCount() / histNDV, nil
}
```

Data Structure Overview

Column Selectivity

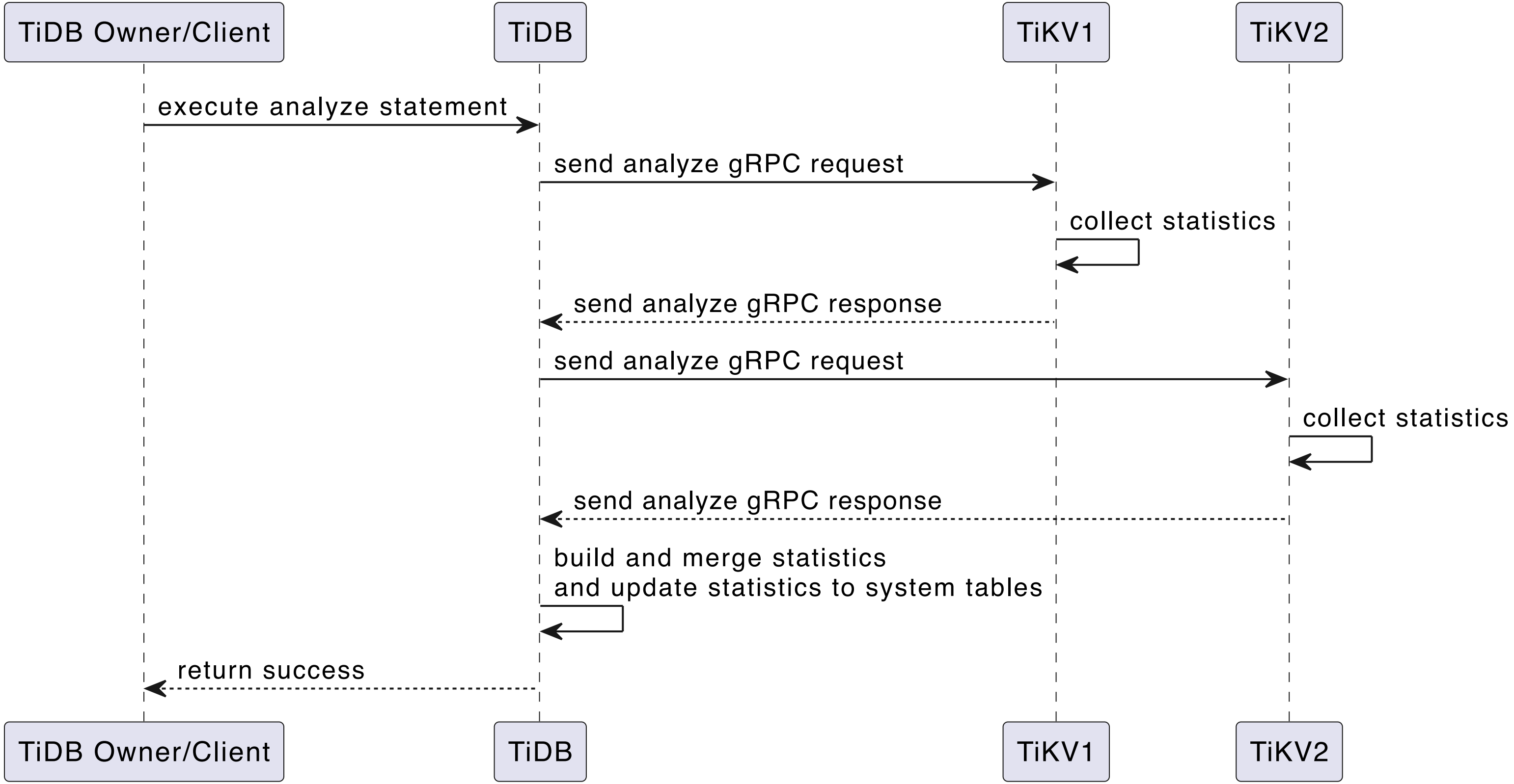
- Not Null Count: The number of not null values in the column.
- NDV: The number of distinct values in the column.

How to calculate the NDV(Non-Distinct Value)?

We use FMSketch(Flajolet-Martin Sketch) to calculate the NDV.

Data Flow Overview

Data Flow Overview


TiDB Owner/Client
TiDB
TiKV1
TiKV2
execute analyze statement
send analyze gRPC request
collect statistics
send analyze gRPC response
send analyze gRPC request
collect statistics
send analyze gRPC response
build and merge statistics
and update statistics to system tables
return success
TiDB Owner/Client
TiDB
TiKV1
TiKV2

Data Structure & Data Flow

TiKV Perspective

In TiKV, we only do two things:

1. Calculate the FMSketch.
2. Sample the data.

Data Structure - FMSketch

TiKV Perspective

Mathematical Assumptions [1]

1. **Independence of Hash Functions**:
   - Assume a good hash function **h(x)** that uniformly distributes input elements over a large range of integers.
2. **Expectation of Trailing Zeros in Hash Values**:
   - For uniformly distributed hash values, the number of trailing zeros in their binary representation follows a geometric distribution.

[1] Flajolet, Philippe; Martin, G. Nigel (1985). "Probabilistic counting algorithms for data base applications"

Data Structure - FMSketch

TiKV Perspective

Algorithm Principles

1. **Hash Mapping**:
   - Map each element of the set to an integer using the hash function h(x).
2. **Trailing Zeros Counting**:
   - For each hash value, count the number of trailing zeros in its binary representation. Record the maximum count **R**.
3. **Cardinality Estimation**:
   - Use the maximum trailing zero count **R** to estimate the cardinality of the set with the formula 2^R.

Flajolet-Martin Sketch

a b c d e f g h i j k l m n o p

Generate Hash Values

Flajolet-Martin Sketch - A Bad Case

a b c d e f g h i j k l m n o jj

Generate Hash Values

Data Structure - Distinct Sampling

TiKV Perspective

Core Principles [1]

1. **Hash Function:**
   - Use a hash function that maps each distinct value to a random `die-level`.
2. **Sample Maintenance:**
   - Maintain a sample S of distinct values and a current level `l`.
3. **Sampling Criterion:**
   - Keep values in S only if their `die-level ≥ l`.
4. **Cardinality Estimation:**
   - Estimate distinct items as $|S| * 2^l$

[1] Phillip B. Gibbons. "Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports"

Data Structure - Distinct Sampling

TiKV Perspective

Algorithm Steps

1. **Initialization:**
   - Start with l = 0 and an empty sample S.
2. **Processing Each Row:**
   - For each row r with target attribute value v:
     - Compute die-level = $h(v)$
     - If die-level ≥ l:
       - Add r to S
3. **Sample Size Control:**
   - If $|S| > k$, increment l and remove items with die-level < l.

Distinct Sampling

Sample Size: 8

a b c d e f g h i j k l m n o jj

Process Next

Die Level: 0

Current Sample Size: 0

Estimated NDV: 0

Data Structure - Distinct Sampling

TiKV Perspective

Estimation and Analysis

1. **Accuracy:**
   - Provides estimates within 0%-10% relative error
   - Much more accurate than previous sampling methods
2. **Efficiency:**
   - Single pass over the data.
   - Only one hash function required.

Data Structure - Bernoulli Sampling

General Perspective

Mathematical Assumptions

1. **Independence of Sample Selection**:
   - Each sample in the data set is selected independently from other samples.
2. **Uniform Sampling Probability**:
   - Each sample is selected with a fixed probability $p(0 \leq p \leq 1)$, uniformly across the entire data set.
3. **Bernoulli Distribution**:
   - Each sample selection follows a Bernoulli distribution with parameter p.

Data Structure - Bernoulli Sampling

General Perspective

Algorithm Principles

1. **Probability Definition**:
   - Define a sampling probability p for selecting each sample.
2. **Independent Sampling**:
   - For each sample in the data set, generate a random number and compare it to p. If the random number is less than p, include the sample in the resulting subset.

Bernoulli Sampling

a b c d e f g h i j k l m n o p

Sample Rate: 0.3

Perform Bernoulli Sampling

Data Structure & Data Flow

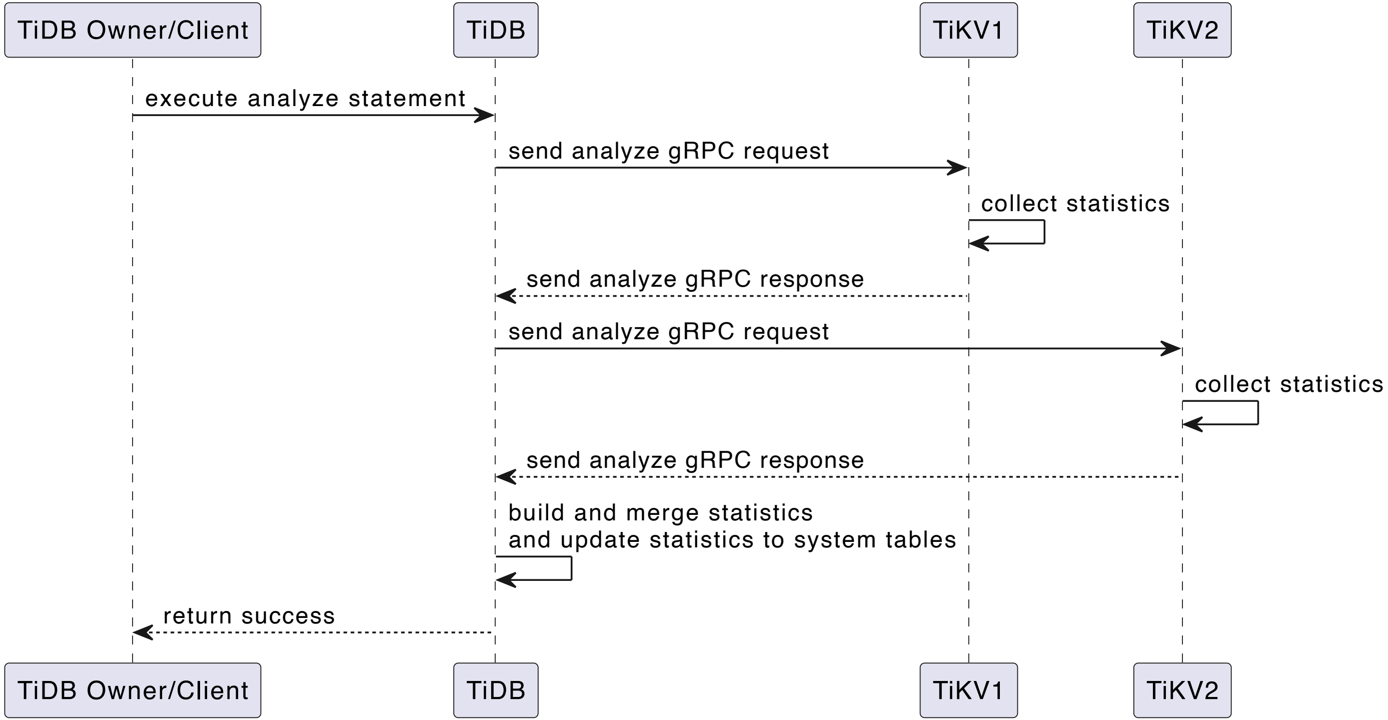
TiDB Perspective

In TiDB, we do the following things:

1. Merge all FMSketches and Sample Data.
2. Build TopN and Histogram.
3. Update statistics to system tables.

Data Structure & Data Flow

Overview

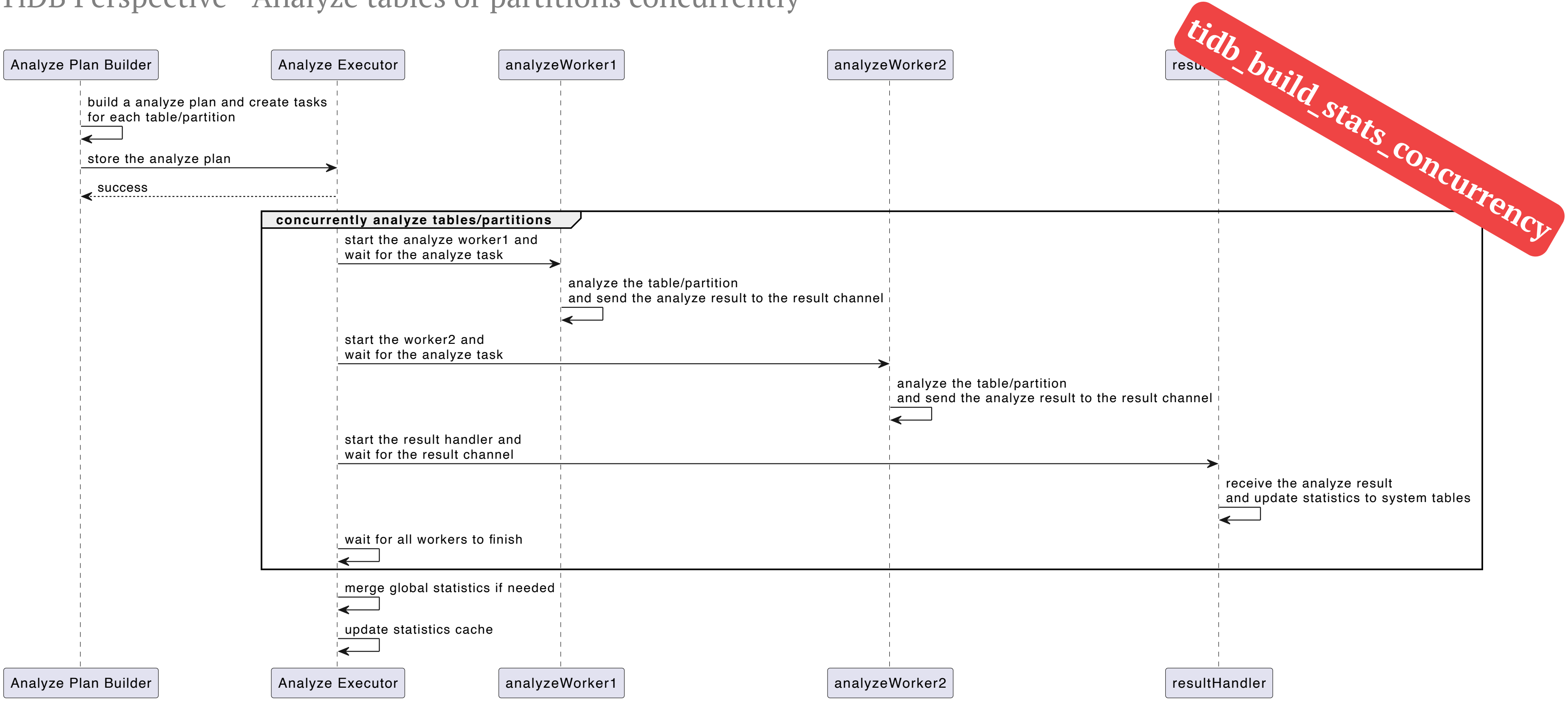

TiDB Owner/Client
TiDB
TiKV1
TiKV2
execute analyze statement
send analyze gRPC request
collect statistics
send analyze gRPC response
send analyze gRPC request
collect statistics
send analyze gRPC response
build and merge statistics
and update statistics to system tables
return success
TiDB Owner/Client
TiDB
TiKV1
TiKV2

🤡Nobody can really master TiDB analyze.jpg

Configuration Name	Description	Default Value	Scope	Affected Component
tidb_build_stats_concurrency	The number of concurrent workers to analyze tables or partitions	2	Global/Session	TiDB + TiKV
tidb_auto_build_stats_concurrency	The number of concurrent workers to automatically analyze tables or partitions	1	Global (only for auto analyze)	TiDB (Owner) + TiKV
tidb_analyze_distsql_scan_concurrency	The number of concurrent workers to scan regions	4	Global/Session	TiKV
tidb_sysproc_scan_concurrency	The number of concurrent workers to scan regions	1	Global (only for auto analyze)	TiKV
tidb_build_sampling_stats_concurrency	1. The number of concurrent workers to merge FMSketches and Sample Data from different regions 2. The number of concurrent workers to build TopN and Histogram	2	Global/Session	TiDB
tidb_analyze_partition_concurrency	The number of concurrent workers to save statistics to the system tables	2	Global/Session	TiDB
tidb_merge_partition_stats_concurrency	The number of concurrent workers to merge global TopN	1	Global/Session	TiDB

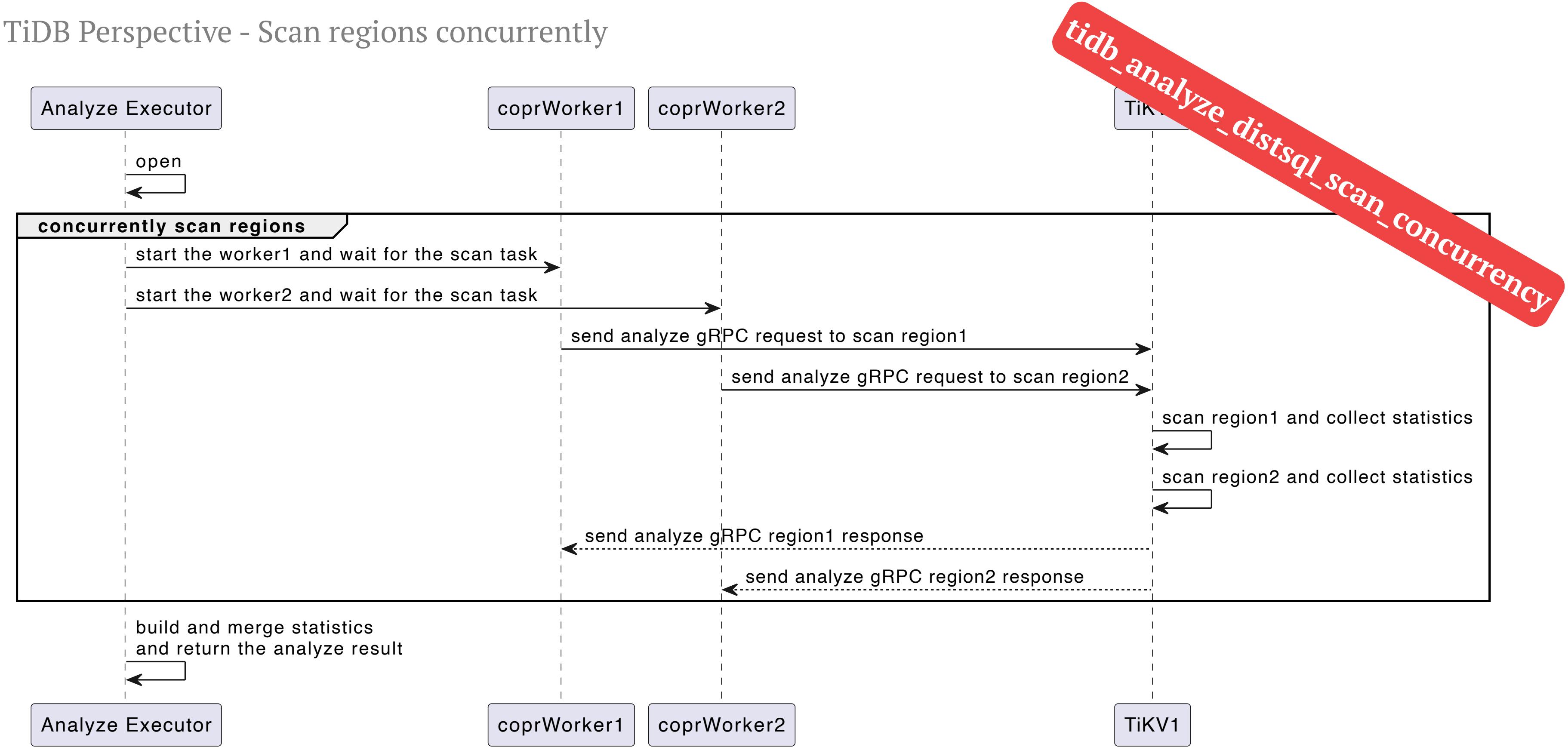
Data Structure & Data Flow

TiDB Perspective - Analyze tables or partitions concurrently


Analyze Plan Builder
Analyze Executor
analyzeWorker1
analyzeWorker2
resu
build a analyze plan and create tasks for each table/partition
store the analyze plan
success
concurrently analyze tables/partitions
start the analyze worker1 and wait for the analyze task
analyze the table/partition and send the analyze result to the result channel
start the worker2 and wait for the analyze task
analyze the table/partition and send the analyze result to the result channel
start the result handler and wait for the result channel
receive the analyze result and update statistics to system tables
wait for all workers to finish
merge global statistics if needed
update statistics cache
Analyze Plan Builder
Analyze Executor
analyzeWorker1
analyzeWorker2
resultHandler
tidb_build_stats_concurrency

Data Structure & Data Flow

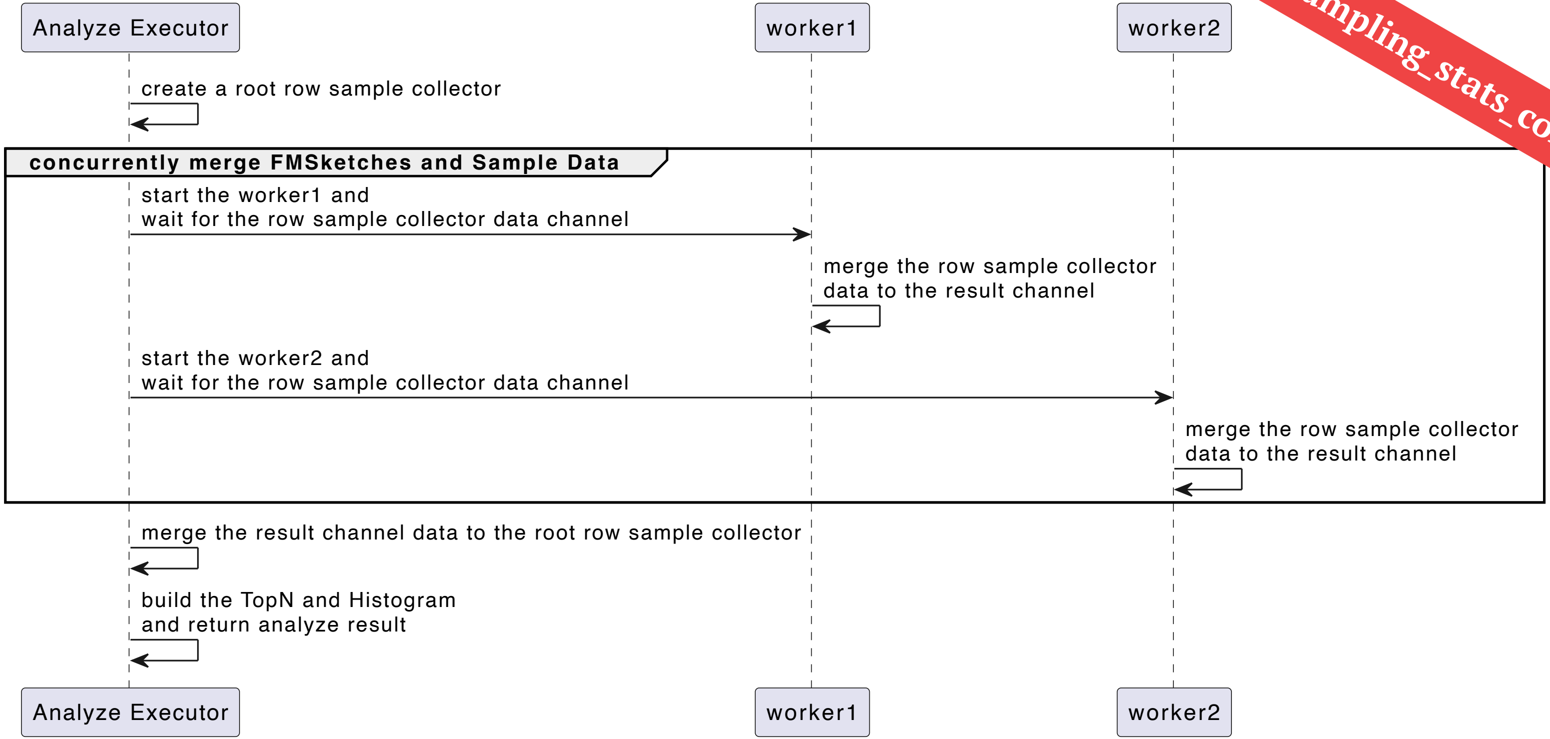
TiDB Perspective - Scan regions concurrently


tidb_analyze_distsql_scan_concurrency
Analyze Executor
coprWorker1
coprWorker2
TiKV1
open
concurrently scan regions
start the worker1 and wait for the scan task
start the worker2 and wait for the scan task
send analyze gRPC request to scan region1
send analyze gRPC request to scan region2
scan region1 and collect statistics
scan region2 and collect statistics
send analyze gRPC region1 response
send analyze gRPC region2 response
build and merge statistics
and return the analyze result
Analyze Executor
coprWorker1
coprWorker2
TiKV1

Data Structure & Data Flow

TiDB Perspective - Merge FMSketches and Sample Data

tidb_build_sampling_stats_concurrency



Analyze Executor

worker1

worker2

create a root row sample collector

concurrently merge FMSketches and Sample Data

start the worker1 and
wait for the row sample collector data channel

merge the row sample collector
data to the result channel

start the worker2 and
wait for the row sample collector data channel

merge the row sample collector
data to the result channel

merge the result channel data to the root row sample collector

build the TopN and Histogram
and return analyze result

Analyze Executor

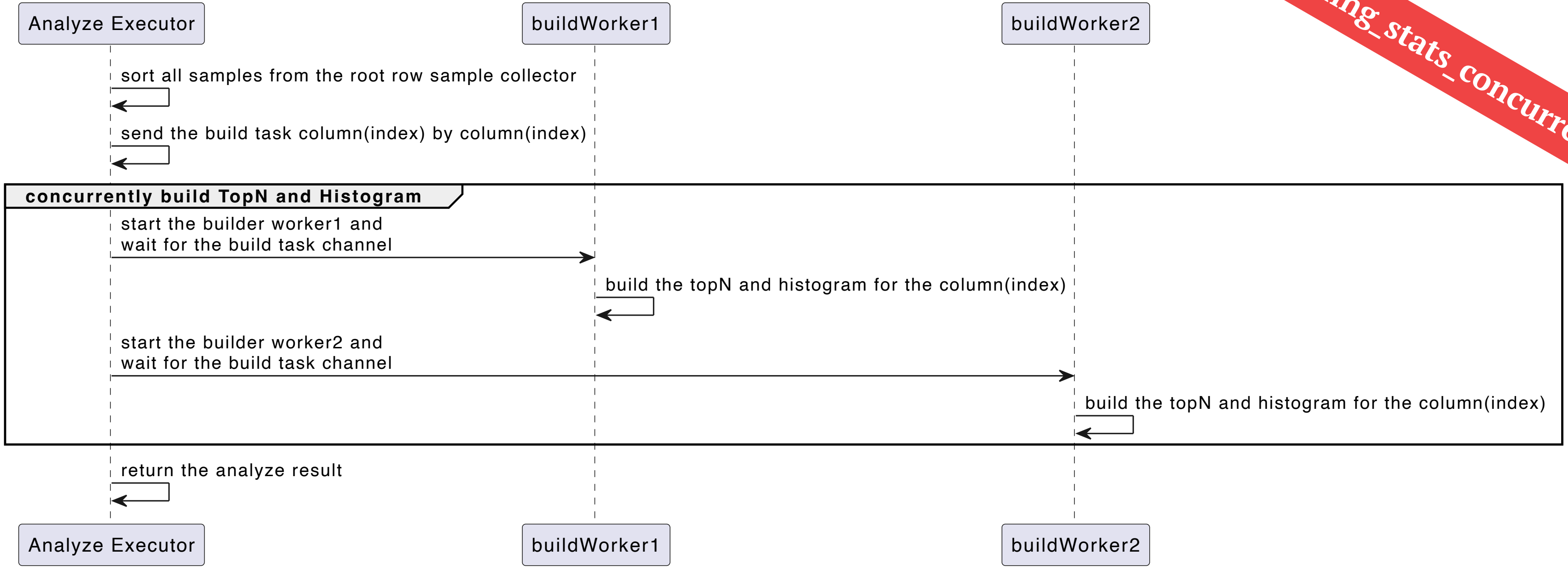
worker1

worker2

Data Structure & Data Flow

TiDB Perspective - Build TopN and Histogram

tidb_build_sampling_stats_concurrency


Analyze Executor
buildWorker1
buildWorker2
sort all samples from the root row sample collector
send the build task column(index) by column(index)
concurrently build TopN and Histogram
start the builder worker1 and
wait for the build task channel
build the topN and histogram for the column(index)
start the builder worker2 and
wait for the build task channel
build the topN and histogram for the column(index)
return the analyze result
Analyze Executor
buildWorker1
buildWorker2

Build TopN

Count: 200 **Sample Length:** 23

Samples

104 23 45 164 57 34 212 92

69 176 116 140 116 128 80 152

188 224 13 200 200 5 5

TopN List

Current:

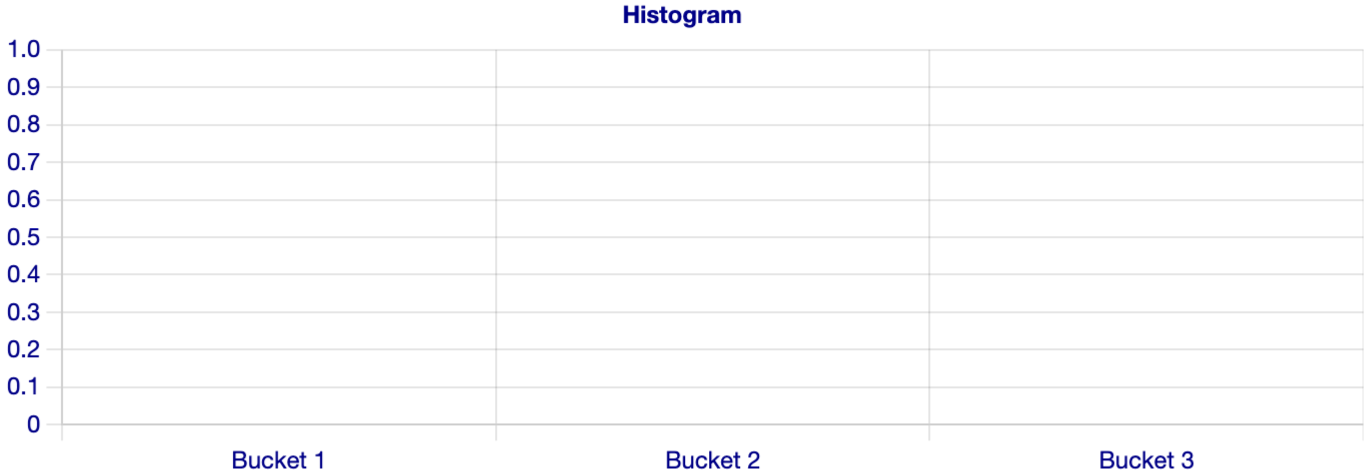
Current Count: 0

Step:

Next Step Reset

Build Histogram

Count: 200 **Sample Length:** 17



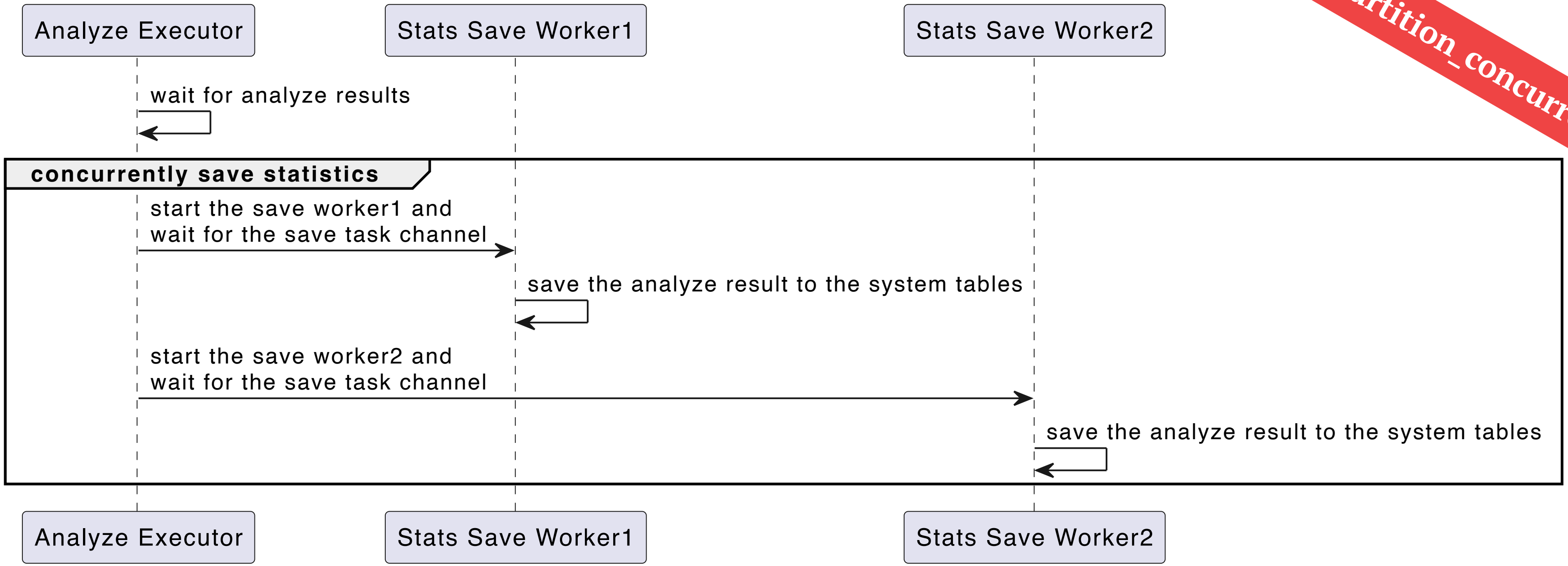
Sample Factor: 11.8 **Per Bucket:** 78.4 **Current Sample:** N/A **Current Bucket:** 1

Step: Click Next Step to start

Data Structure & Data Flow

TiDB Perspective - Save Statistics

tidb_analyze_partition_concurrency



Analyze Executor

Stats Save Worker1

Stats Save Worker2

wait for analyze results

concurrently save statistics

start the save worker1 and
wait for the save task channel

save the analyze result to the system tables

start the save worker2 and
wait for the save task channel

save the analyze result to the system tables

Analyze Executor

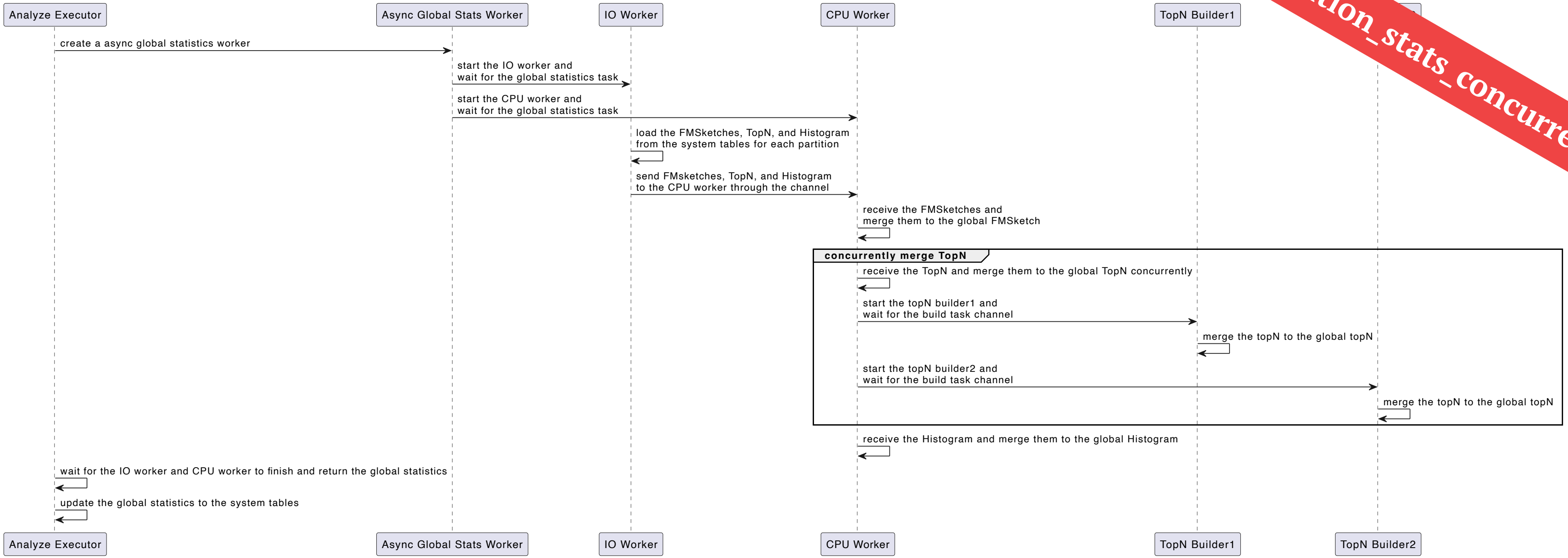
Stats Save Worker1

Stats Save Worker2

Data Structure & Data Flow

TiDB Perspective - Merge Global Statistics

tidb_merge_partition_stats_concurrency



Analyze Executor | Async Global Stats Worker | IO Worker | CPU Worker | TopN Builder1 | TopN Builder2

create a async global statistics worker

start the IO worker and
wait for the global statistics task

start the CPU worker and
wait for the global statistics task

load the FMSketches, TopN, and Histogram
from the system tables for each partition

send FMsketches, TopN, and Histogram
to the CPU worker through the channel

receive the FMSketches and
merge them to the global FMSketch

concurrently merge TopN

receive the TopN and merge them to the global TopN concurrently

start the topN builder1 and
wait for the build task channel

merge the topN to the global topN

start the topN builder2 and
wait for the build task channel

merge the topN to the global topN

receive the Histogram and merge them to the global Histogram

wait for the IO worker and CPU worker to finish and return the global statistics

update the global statistics to the system tables

Analyze Executor | Async Global Stats Worker | IO Worker | CPU Worker | TopN Builder1 | TopN Builder2

Improve the Analyze

What can we do?

Tracking Document

Statistics Project Planning and Implementation

Statistics Tech Debt

Blogs

I started a new series blog post to discuss the Analyze feature improvement.

NCRMTA1: Surprise analyze-partition-concurrency-quota

NCRMTA2: Accelerate Auto-Analyze of Partitioned Tables

Q&A

Do you have any questions?

Thank You!