TiDB 可观测性的设计与实现

– 陈霜

InfoQ

精彩继续！
更多一线大厂前沿技术案例

广州站

QCon
全球软件开发大会

时间：2022年7月31-8月1日
地点：广州·万富希尔顿酒店

扫码查看大会详情>>

北京站

GMTC
全球大前端技术大会

时间：2022年8月
地点：北京·国际会议中心

扫码查看大会详情>>

北京站

QCon
全球软件开发大会

时间：2022年9月
地点：北京·国际会议中心

扫码查看大会详情>>

About me

- 陈霜, TiDB Insight R&G Engineer, PingCAP
- chenshuang@pingcap.com
- Github: crazycs520

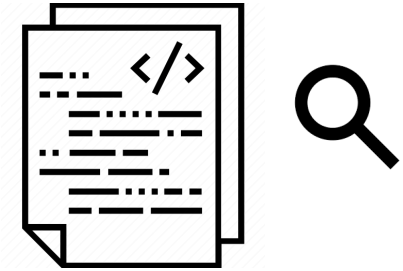
Agenda

- TopSQL: Bind SQL With CPU Resources
- System Table
  - SQL Statements Implementation
  - SLOW QUERY Implementation
- Continuous Profiling Implementation

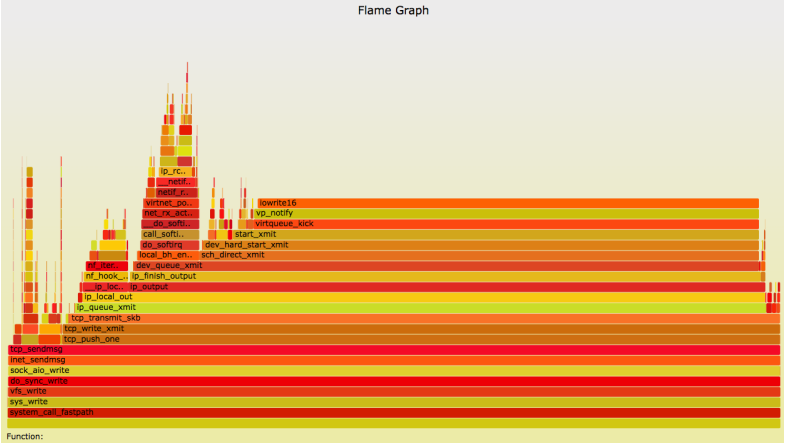
TopSQL: Bind SQL With CPU Resources

Background

- Database Performance is getting worse!
  - QPS is dropping
  - Latency is rising



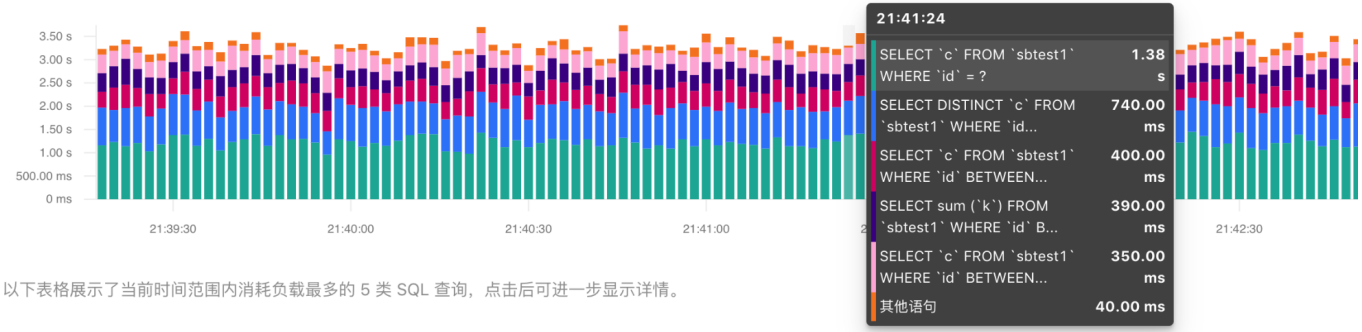
- What OS tells us?
  - **top / htop**
  - High CPU usage



- Which line of code is running?
  - **CPU Profiler**

TopSQL

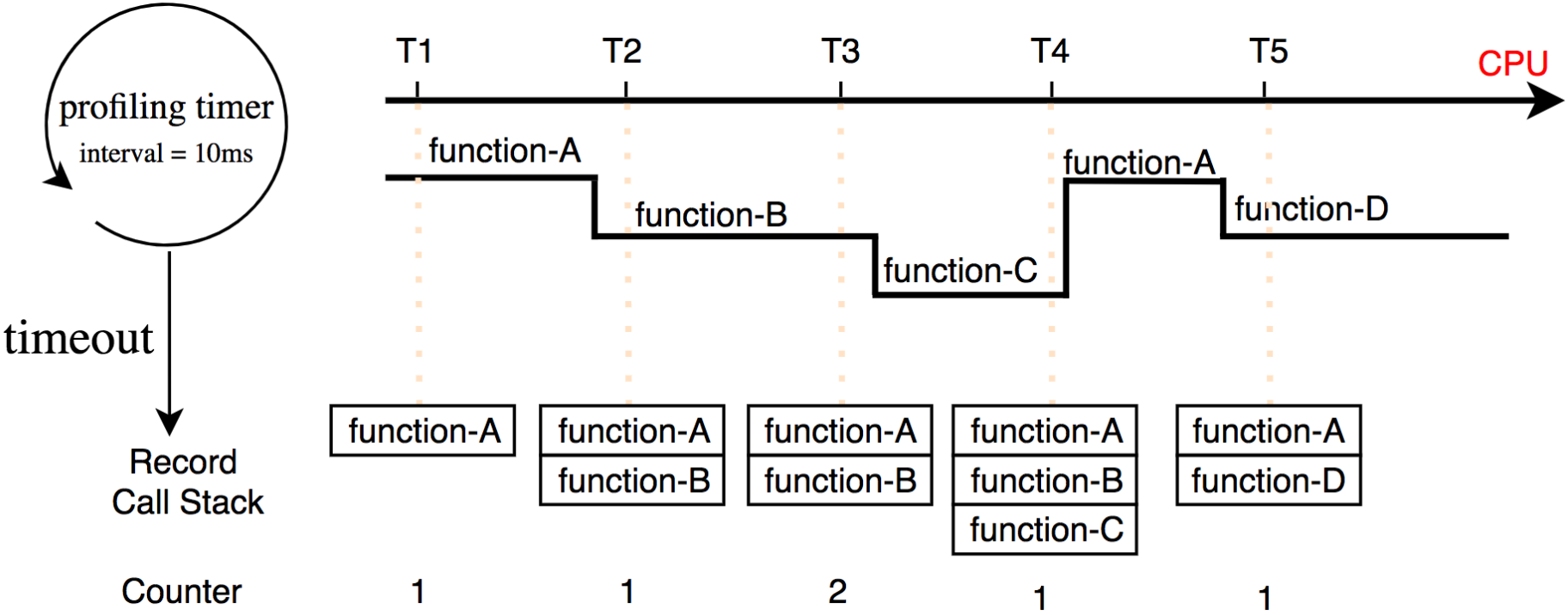
tidb - 172.31.9.146:1... | 最近 5 min | 刷新



以下表格展示了当前时间范围内消耗负载最多的 5 类 SQL 查询，点击后可进一步显示详情。

CPU	SQL 语句
3.10 min	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` = ?
1.93 min	`SELECT DISTINCT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ? `ORDER BY…`
1.11 min	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ? `ORDER BY` `c`
57.69 s	`SELECT` `sum` (`k`) `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ?
54.70 s	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ?
19.54 s	*其他语句*

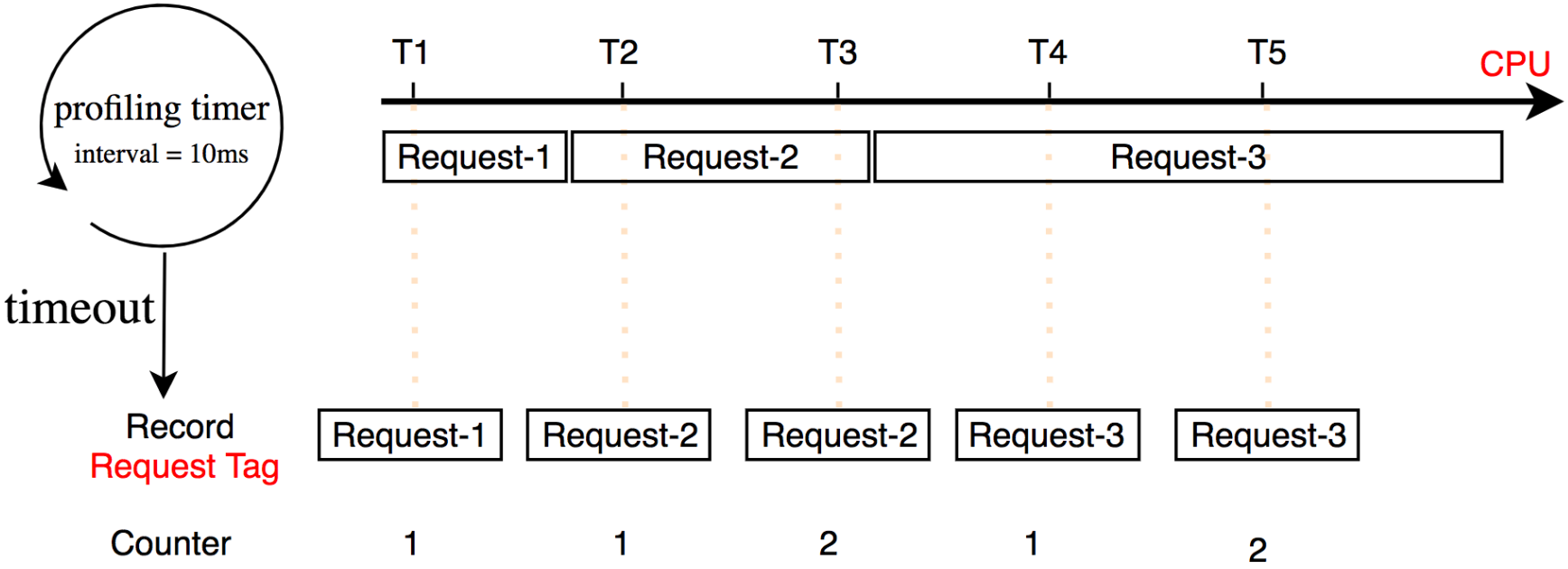
How CPU Profiling works: Sampling



How CPU Profiler works: Accumulation

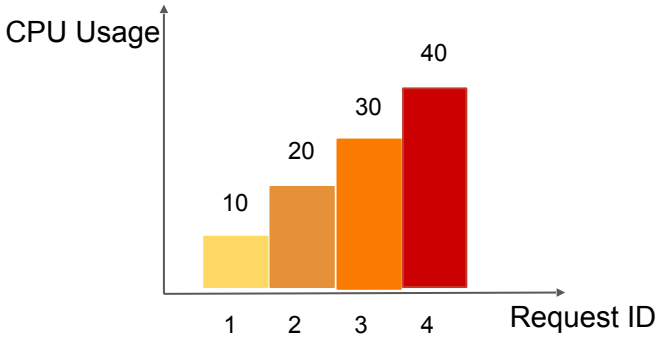
Stack	Count
A	1
A -> B	2
A -> B -> C	1
A -> D	1

CPU Profiling By SQL Request



CPU Profiling By SQL Request

Request tag	Count
Request_ 1	10
Request_ 2	20
Request_ 3	30
Request_ 4	40



CPU Profiling By SQL Request

- rust-demo
- go-demo

CPU Profiling By SQL Request

- rust-demo
- go-demo

点击以添加演讲者备注

TopSQL: SQL Tag - Digest

Original SQL

```
select id from t where a=1;
select id from t where a=2;
```

Normalization

Normalized SQL

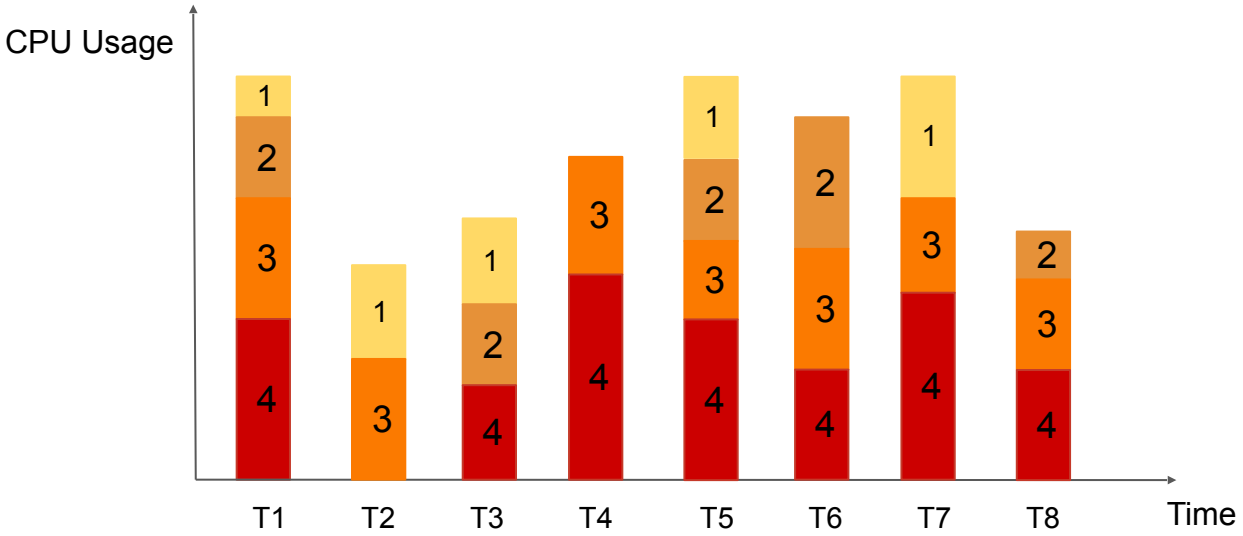
```
select id from t where a=?;
```

Hash

SQL Digest

```
e27d8aa363129bf86cc3b35eb7a042c770802a539716e20ea2af30bd924
e5909
```

TopSQL



1 SELECT * FROM t1 WHERE id = ?

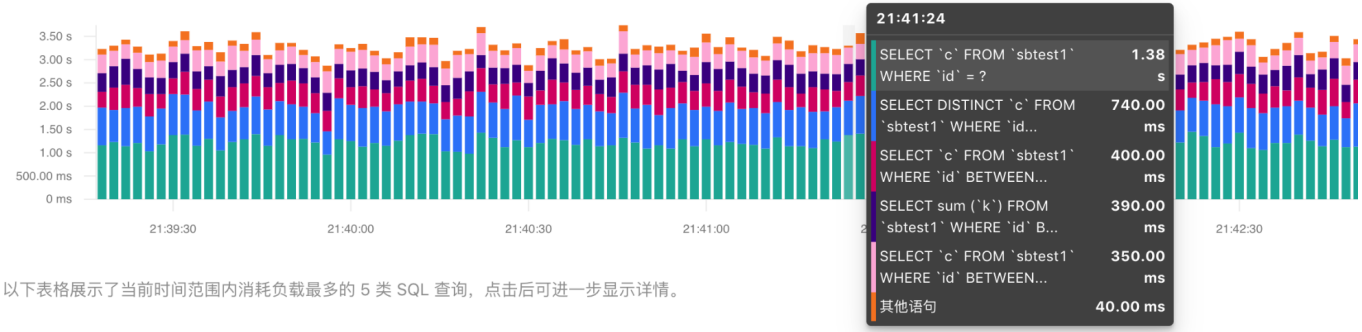
2 UPDATE t2 SET a=a+1 WHERE id = ?

3 DELETE FROM t3 WHERE id = ?

4 SELECT COUNT(*) FROM t1

TopSQL

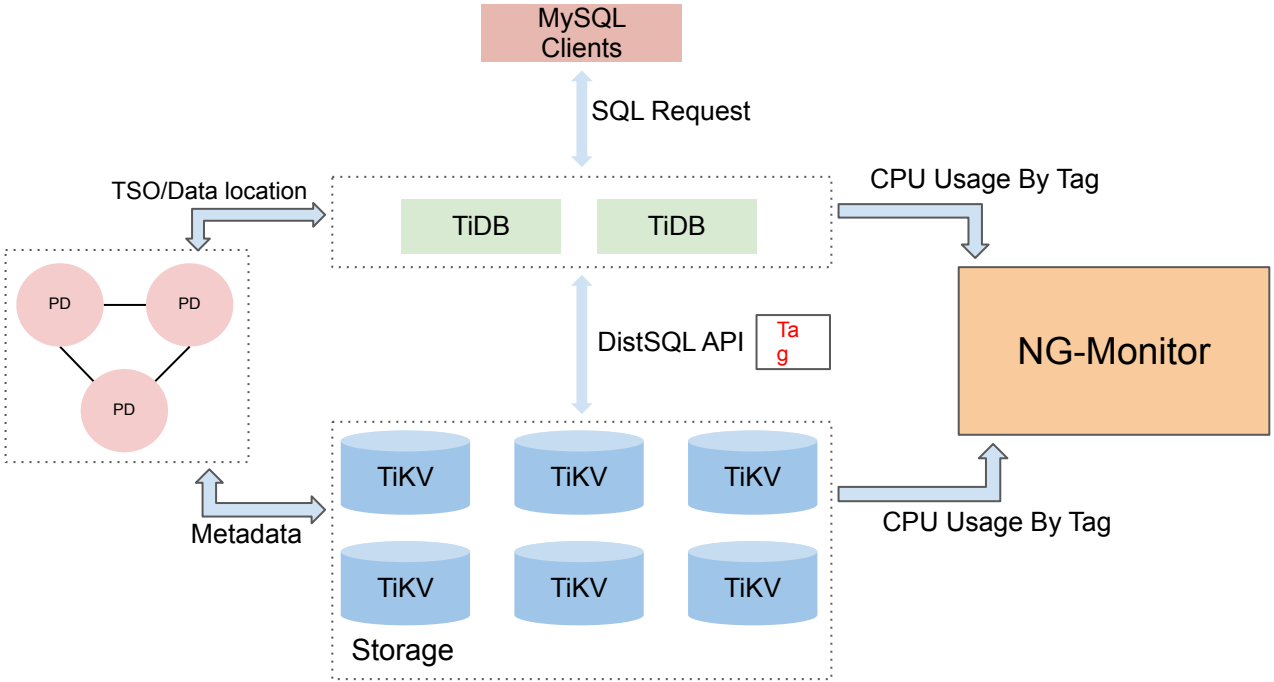
tidb - 172.31.9.146:1... | 最近 5 min | 刷新



以下表格展示了当前时间范围内消耗负载最多的 5 类 SQL 查询，点击后可进一步显示详情。

CPU	SQL 语句
3.10 min	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` = ?
1.93 min	`SELECT DISTINCT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ? `ORDER BY`…
1.11 min	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ? `ORDER BY` `c`
57.69 s	`SELECT` `sum` (`k`) `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ?
54.70 s	`SELECT` `c` `FROM` `sbtest1` `WHERE` `id` `BETWEEN` ? `AND` ?
19.54 s	*其他语句*

TopSQL


MySQL Clients
SQL Request
TSO/Data location
TiDB
TiDB
CPU Usage By Tag
PD
PD
PD
DistSQL API
Ta g
NG-Monitor
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
Metadata
CPU Usage By Tag
Storage

Another approach to CPU resource bind in Go

Goroutine CPU Stats

- Try to collect goroutine runtime information.

```
begin := GetGoroutineStats()

executeSQL()

end := GetGoroutineStats()

queryCost := end.Sub(begin) // 获取执行 SQL 消耗的 cpu stats 信息
```

Trace

- Use Go trace to collect runtime information.

```
curl http://localhost:10080/debug/pprof/trace\?seconds\=1 --output trace.out
go tool trace trace.out
```

- In Goroutine analysis Page, chose the Goroutine you want to view.

Goroutine Name: github.com/pingcap/tidb/server.(*Server).onConn
Number of Goroutines: 1
Execution Time: 95.33% of total program execution time
Network Wait Time: graph(download)
Sync Block Time: graph(download)
Blocking Syscall Time: graph(download)
Scheduler Wait Time: graph(download)

Goroutine	Total		Execution	Network wait	Sync block	Blocking syscall	Scheduler wait	GC sweeping	GC pause
15887	1000ms		995ms	2818μs	0ns	41μs	1341μs	3113μs (0.3%)	1760μs (0.2%)

Trace - drawback

- Large performance impact.
- May generate a lot of trace data, and then parsing this data is also consume a lot of time.

How Go trace works?

Time(ms)	Event	goroutine_id	description
0	GoCreate	1	创建一个 goroutine
10	GoStart	1	开始运行 goroutine
30	GoBlockSelect	1	被 block 了，暂停运行 goroutine
50	GoUnpark	1	goroutine 没有被 block 了
60	GoStart	1	开始运行 goroutine
80	GoEnd	1	结束 goroutine

How Go trace works?

Time(ms)	Event	goroutine_id	description
0	GoCreate	1	创建一个 goroutine
10	GoStart	1	开始运行 goroutine
30	GoBlockSelect	1	被 block 了，暂停运行 goroutine
50	GoUnpark	1	goroutine 没有被 block 了
60	GoStart	1	开始运行 goroutine
80	GoEnd	1	结束 goroutine

Type	**Duration**	**Calculate**
total_time	80ms	80-0
exec_time	40ms	(30-10) + (80-60)
sync_block_time	20ms	50 - 30
scheduler_wait_time	20ms	(10-0) + (60-50)

Modify Go runtime to collect Goroutine stats

```go
type g struct {
    goid int64 // goroutine id.
    . . .
    stats GStats // goroutine runtime stats
}

type GStats struct {
     creationTime int64
     endTime      int64

            lastStartTime    int64
            blockSchedTime   int64
     ...

            execTime       int64
            schedWaitTime int64
     ...
}
```

```go
            // 创建 goroutine 时
            traceGoCreate(newg, newg.startpc)
            newg.stats.recordGoCreate()

            // 开始执行 goroutine 时
            traceGoStart()
            gp.stats.recordGoStart()


            // goroutine 被抢占而停止运行时
            traceGoPreempt()
            gp.stats.recordGoSched()

        // 其他的埋点代码 …
        ...
```

Modify Go runtime to collect Goroutine stats

- demo: crazycs520/go

```
# build
git clone -b stats-dev2 https://github.com/crazycs520/go.git
cd go/src
./make.bash
cd ..
export GOROOT=$PWD
export PATH=$GOROOT/bin:$PATH

# demo
$ cd example
$ go run goroutine_stats.go
total: 1.054196173s, exec: 366.938307ms, network_wait: 131.648319ms, sync_block:
443.454523ms, block_syscall: 0s, scheduler_wait: 148.731µs, gc_sweep: 7.82µs
```

Modify Go runtime to collect Goroutine stats

- demo: crazycs520/go

```
# build
git clone -b stats-dev2 https://github.com/crazycs520/go.git
cd go/src
./make.bash
cd ..
export GOROOT=$PWD
export PATH=$GOROOT/bin:$PATH

# demo
$ cd example
$ go run goroutine_stats.go
total: 1.054196173s, exec: 366.938307ms, network_wait: 131.648319ms, sync_block: 443.454523ms,
block_syscall: 0s, scheduler_wait: 148.731µs, gc_sweep: 7.82µs
```

Modify Go runtime to collect Goroutine stats

- Drawback
  - Need to maintain a go branch by ourself.

- Related Issue:
  - proposal: runtime: add per-goroutine CPU stats · Issue #41554

System Table

SQL Statements Implementation

PingCAP

TiDB Dashboard
PD v5.5.0-alpha

- Overview
- Cluster Info
- Top SQL
- SQL Statements
- Slow Queries
- Key Visualizer
- Cluster Diagnostics
- Search Logs
- Advanced Debugging
  - Profiling Instances
    - Manual Profiling
    - Continuous Profiling
  - Debug Data

Recent 30 min | All Databases | All Kinds

Statement Template	Total Latency	Mean Latency	# Exec	Mean Memory
SELECT `c` FROM `sbtest1` WHERE `id` = ?	20.2 min	2.8 ms	439.2 K	0 B
COMMIT	7.9 min	10.8 ms	43.9 K	0 B
SELECT DISTINCT `c` FROM `sbtest1` WHERE `id` BETWEEN ? AND…	3.8 min	5.1 ms	43.9 K	111.9 KiB
SELECT `c` FROM `sbtest1` WHERE `id` BETWEEN ? AND ? ORDER …	3.1 min	4.2 ms	43.9 K	27.6 KiB
SELECT `c` FROM `sbtest1` WHERE `id` BETWEEN ? AND ?	3.0 min	4.1 ms	43.9 K	14.6 KiB
SELECT sum (`k`) FROM `sbtest1` WHERE `id` BETWEEN ? AND ?	3.0 min	4.1 ms	43.9 K	1.7 KiB
DELETE FROM `sbtest1` WHERE `id` = ?	2.4 min	16.0 ms	8.9 K	712.0 B
UPDATE `sbtest1` SET `c` = ? WHERE `id` = ?	2.4 min	16.0 ms	8.9 K	676.0 B
UPDATE `sbtest1` SET `k` = `k` + ? WHERE `id` = ?	2.1 min	14.4 ms	8.9 K	676.0 B
BEGIN	1.6 min	2.1 ms	43.9 K	0 B
INSERT INTO `sbtest1` (`id`, `k`, `c`, `pad`) VALUES (...)	2.4 s	266.2 µs	8.9 K	700.0 B
SET @@global.tidb_enable_top_sql = ?	181.4 ms	181.4 ms	1	0 B
SELECT DISTINCT `floor` (`unix_timestamp` (`summary_begin_t…	51.7 ms	17.2 ms	3	30.1 KiB
SELECT DISTINCT `stmt_type` FROM `information_schema`.`clus…	38.3 ms	12.8 ms	3	11.3 KiB

STATEMENTS_SUMMARY

- Groups the SQL statements by the SQL digest and the plan digest, and provides statistics for each SQL category.

- Is a system **memory** table in INFORMATION_SCHEMA
  - LRU Cache
  - At most store 3000 kinds of SQL statements. Control by tidb_stmt_summary_max_stmt_count
- Is periodically cleared, and only recent (30min by default) aggregated results are retained and displayed.
  - tidb_stmt_summary_refresh_interval

- **STATEMENTS_SUMMARY_HISTORY** saves the historical data of STATEMENTS_SUMMARY
  - tidb_stmt_summary_history_size

Slow Query Implementation

PingCAP

TiDB Dashboard
PD v5.5.0-alpha

- Overview
- Cluster Info
- Top SQL
- SQL Statements
- Slow Queries
- Key Visualizer
- Cluster Diagnostics
- Search Logs
- Advanced Debugging
  - Profiling Instances
    - Manual Profiling
    - Continuous Profiling
  - Debug Data

Recent 30 min | All Databases | Limit 100

Query	Finish Time	Latency	Parse Time	Generate Plan Time
SELECT c FROM sbtest1 WHERE id = ? [arguments: 625488];	Today at 6:40 PM (GM...	54.7 ms	0 ns	11.9 µs
SELECT c FROM sbtest2 WHERE id = ? [arguments: 504827];	Today at 6:40 PM (GM...	59.1 ms	0 ns	27.0 µs
SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c […	Today at 6:40 PM (GM...	52.8 ms	0 ns	103.1 µs
SELECT SUM(k) FROM sbtest1 WHERE id BETWEEN ? AND ? [argume…	Today at 6:40 PM (GM...	54.8 ms	0 ns	191.3 µs
SELECT c FROM sbtest1 WHERE id = ? [arguments: 501451];	Today at 6:40 PM (GM...	53.2 ms	0 ns	18.6 µs
SELECT c FROM sbtest2 WHERE id BETWEEN ? AND ? ORDER BY c […	Today at 6:40 PM (GM...	54.3 ms	0 ns	223.6 µs
SELECT c FROM sbtest2 WHERE id = ? [arguments: 524818];	Today at 6:40 PM (GM...	59.7 ms	0 ns	22.4 µs
SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? [arguments: …	Today at 6:40 PM (GM...	70.4 ms	0 ns	94.4 µs
SELECT c FROM sbtest2 WHERE id = ? [arguments: 497998];	Today at 6:40 PM (GM...	60.6 ms	0 ns	10.9 µs
SELECT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORDER BY c […	Today at 6:40 PM (GM...	51.4 ms	0 ns	222.6 µs
SELECT DISTINCT c FROM sbtest1 WHERE id BETWEEN ? AND ? ORD…	Today at 6:40 PM (GM...	78.0 ms	0 ns	607.8 µs
SELECT c FROM sbtest2 WHERE id = ? [arguments: 499820];	Today at 6:40 PM (GM...	54.2 ms	0 ns	12.8 µs
SELECT DISTINCT c FROM sbtest2 WHERE id BETWEEN ? AND ? ORD…	Today at 6:40 PM (GM...	52.3 ms	0 ns	117.3 µs

SLOW_QUERY

- Is a system table in INFORMATION_SCHEMA
- The table data is parsed from slow-query-file.
- Use SQL to query **SLOW_QUERY** instead of grep slow-query-file.

SLOW QUERY FILE

The slow-query-file consists of SQL statements that take more than tidb_slow_log_threshold milliseconds to execute.

An example record in slow-query-file :

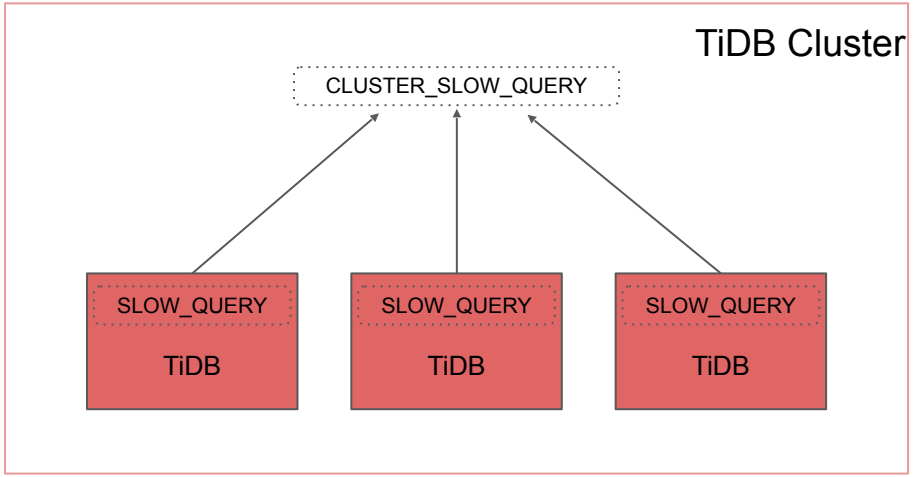
```
# Time: 2019-08-14T09:26:59.487776265+08:00
# Txn_start_ts: 410450924122144769
# User: root@127.0.0.1
# Conn_ID: 3086
# Query_time: 1.527627037
# Parse_time: 0.000054933
# Compile_time: 0.000129729
# Cop_time: 0.387091637 Process_time: 0.613 Request_count: 2 Total_keys: 131073 Process_keys: 131072 Prewrite_time: 0.335415029 Commit_time: 0.032175429
Get_commit_ts_time: 0.000177098 Local_latch_wait_time: 0.106869448 Write_keys: 131072 Write_size: 3538944 Prewrite_region: 1
# DB: test
# Is_internal: false
# Digest: 50a2e32d2abbd6c1764b1b7f2058d428ef2712b029282b776beb9506a365c0f1
# Stats: t:pseudo
# Num_cop_tasks: 2
# Cop_proc_avg: 0.3065 Cop_proc_p90: 0.322 Cop_proc_max: 0.322 Cop_proc_addr: 127.0.0.1:20160
# Mem_max: 525211
# Prepared: false
# Plan_from_cache: false
# Succ: true
# Plan:
tidb_decode_plan('ZJAwCTMyXzcJMAkyMAlkYXRhOlRhYmxlU2Nhbl82CjEJMTBfNgkxAR0AdAEY1Dp0LCByYW5nZTpbLWluZiwraW5mXSwga2VlcCBvcmRlcjpmYWxzZSwgc3RhdHM6cHNldWRvCg==')
insert into t select * from t;
```

The format is same with mysql slow query log,so tools like pt-query-digest also work.

CLUSTER TABLES

STATEMENTS_SUMMARY/SLOW_QUERY only contains the data of current tidb-server.

CLUSTER_STATEMENTS_SUMMARY/**CLUSTER**_SLOW_QUERY are system table of TiDB, It contains the related data of all tidb-servers.



Continuous Profiling

Continuous Profiling

PingCAP

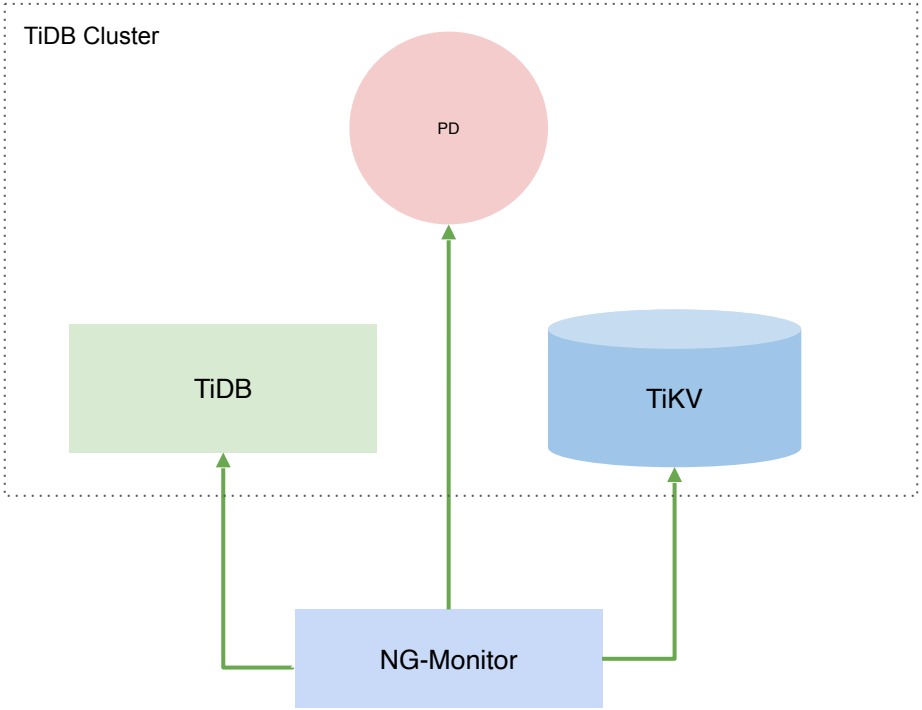
TiDB Dashboard
PD v5.5.0-alpha

- Overview
- Cluster Info
- Top SQL
- SQL Statements
- Slow Queries
- Key Visualizer
- Cluster Diagnostics
- Search Logs
- Advanced Debugging
  - Profiling Instances
    - Manual Profiling
    - Continuous Profiling
  - Debug Data

Range End Time: Select date Range Duration: -2h Query

Instances	Status	Start At	Duration (sec)
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:25:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:24:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:23:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:22:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:21:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:20:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:19:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:18:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:17:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:16:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:15:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:14:01 PM (GMT+8)	10
1 TiKV, 1 TiDB, 1 PD, 1 TiFlash	Finished	Feb 24, 2022 7:13:01 PM (GMT+8)	10

Continuous Profiling



TGO 鲲鹏会 – 助力企业赢得数字化先机

Geekbang. 极客邦科技

TGO 鲲鹏會 tgo.infoq.cn

TGO 鲲鹏会是科技领导者同侪学习社区，致力于把技术领导者和专家连接在一起，通过领导力峰会、专属小组活动、闭门沙龙等形式，为所有“孤军奋战”的科技管理者获得自身的成长和职业的发展。

TGO鲲鹏会目前拥有1600+位高端会员。在北京、上海、深圳、广州、杭州、南京、成都、厦门、台北、硅谷、武汉、苏州等全球十二个城市设立分会。

Q&A