TiDB

浅谈全链路监控：从应用到数据库到 Runtime

黄东旭, Co-founder & CTO, PingCAP

关于我

黄东旭，联合创始人 & CTO @ PingCAP

做分布式数据库的程序员
- 现在能写代码的时间是奢侈品

TiDB 的亲爹之一兼首席客服和新功能的第一个用户
- 冤有头债有主，SQL 慢了来找我。。。

偶尔玩玩音乐
- 摇滚乐->实验音乐

Go 的粉丝！！！！

关于 PingCAP

一个有趣的公司

信仰开源，做分布式数据库的，叫 TiDB（好像还蛮火的）

TiDB 是分布式的，PingCAP 这个公司也是分布式的（北京、上海、广州、深圳、成都、杭州、新加坡、东京、硅谷）

We're hiring!

今天的演讲内容来自我最近的一个真实经历。。。。

一个真实的故事

老板：怎么应用那么卡？

前端开发：是不是你网络不行

老板：换了个网还是一样

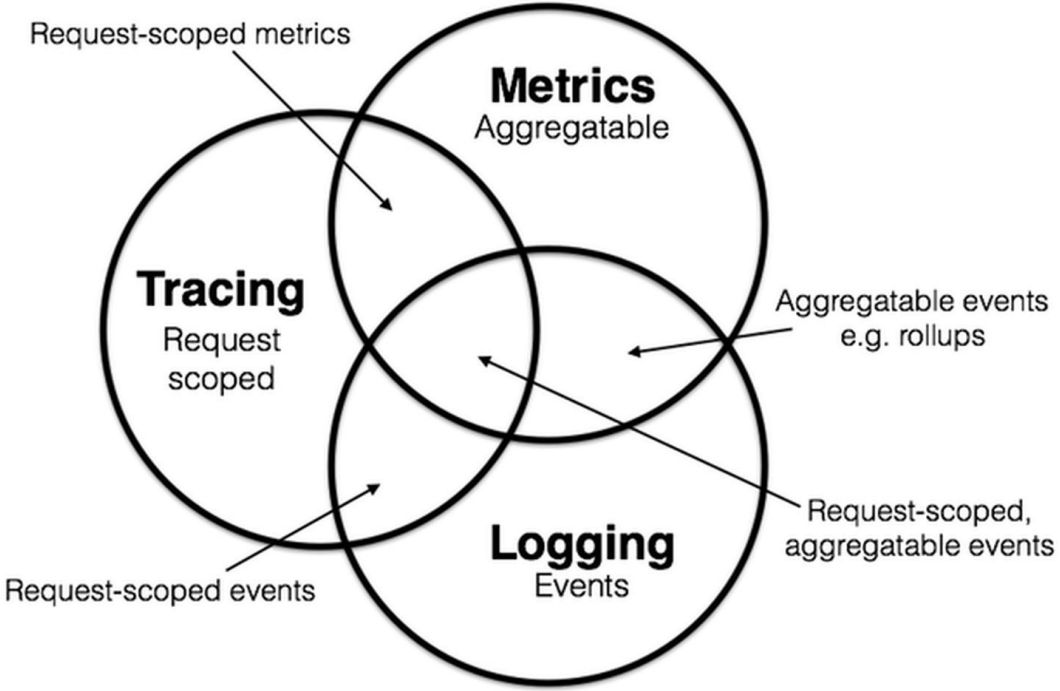
前端开发：找到原因了，这个 REST API 卡了几秒返回

后端开发：这个 API 没啥逻辑，肯定是 TiDB 的问题

DBA：没有啊，我看数据库响应时间也就几 ms，肯定是你的应用写得💩

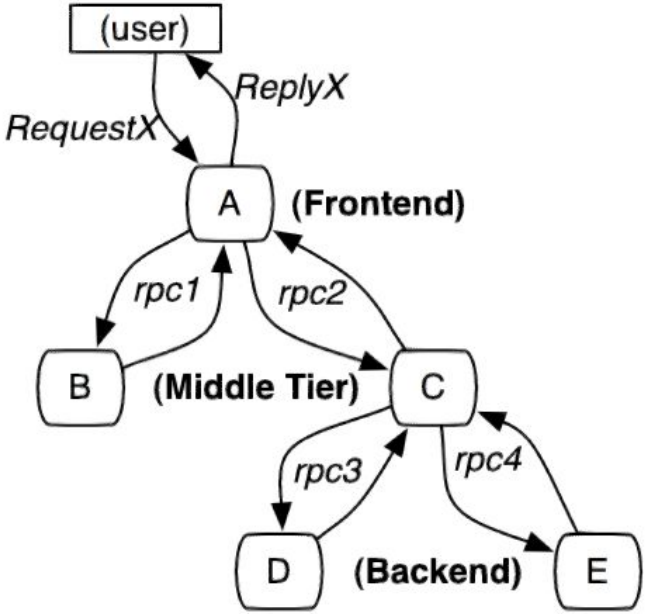
后端开发：....（事后一看果然是）

Logging / Tracing / Metrics



分布式 Tracing 面临的挑战

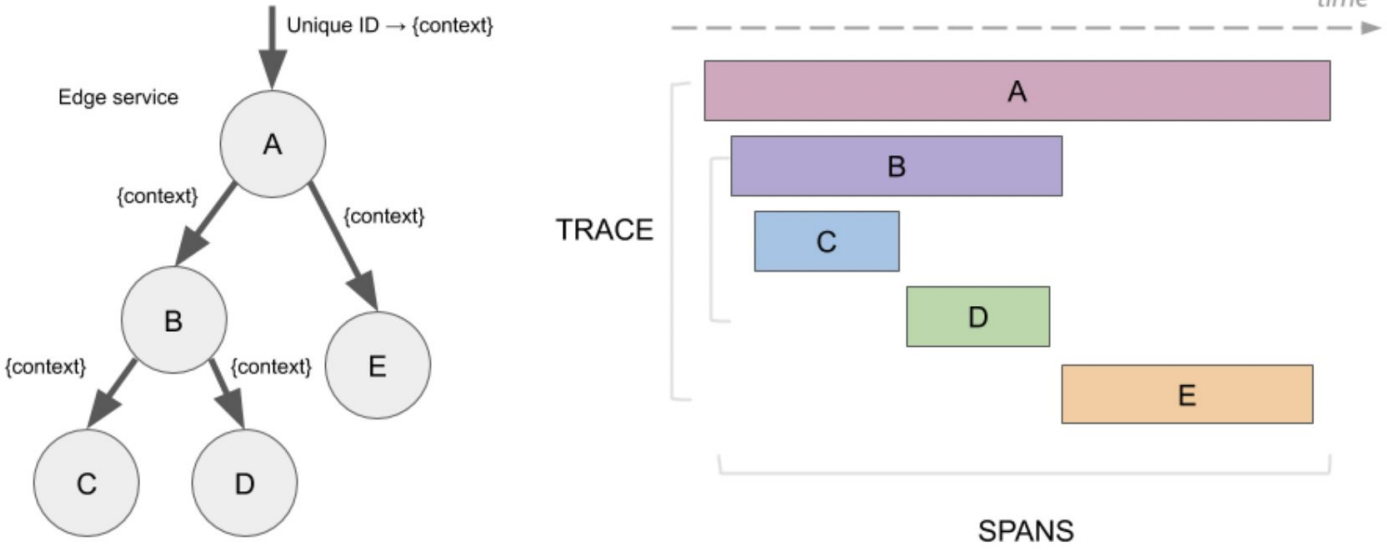
- 领域知识和业务强相关
  - 工程师不会熟悉每个模块
- 跨越多团队 / 语言 / 服务 / 模块 / 物理机
- 业务侵入性和性能损耗
- 实时，秒级别有效性



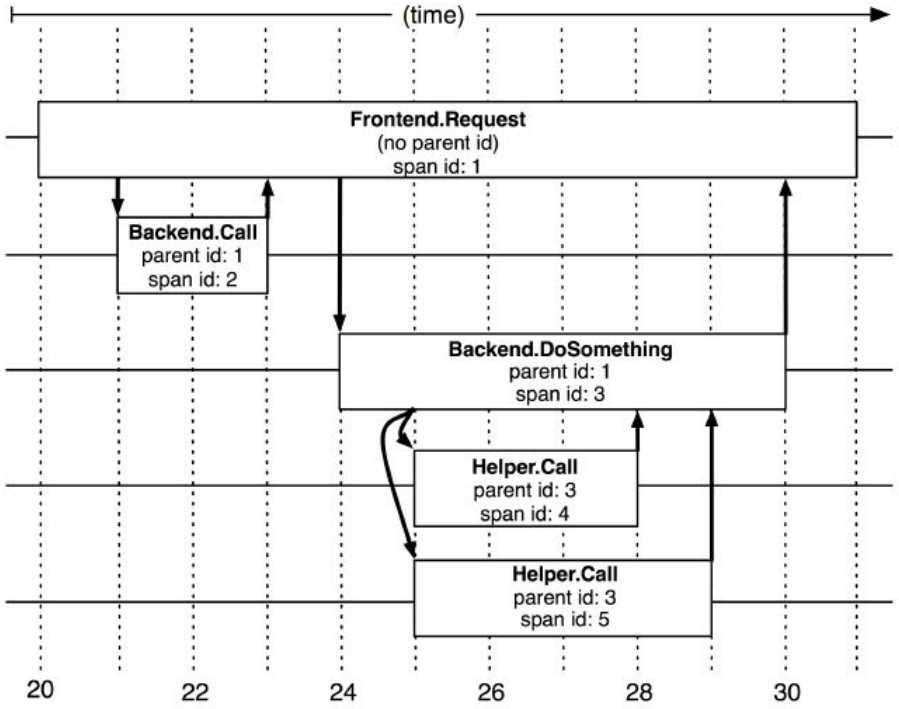
行业现状

- 起点：《Dapper, a Large-Scale Distributed Systems Tracing Infrastructure》 - Google 2010
- OpenTracing
  - Zipkin
  - Uber Jaeger
- OpenCensus
- OpenTracing + OpenCensus = OpenTelemetry

基本概念：Trace & Span

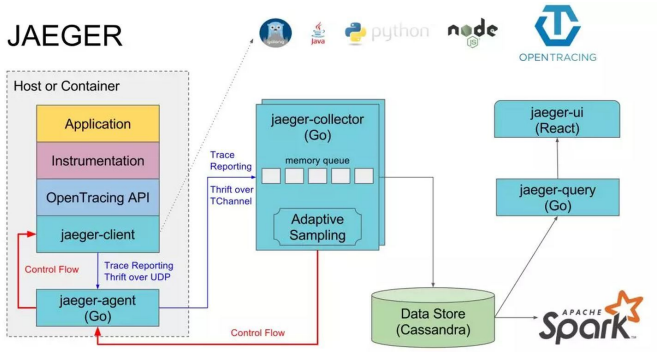


Trace & Span 例子



Jaeger

- CNCF 毕业项目
- 实现 OpenTracing 标准
- UI 友好
- 多种 Sampling 策略
- 后端存储分离
- Go! (图标也好看)

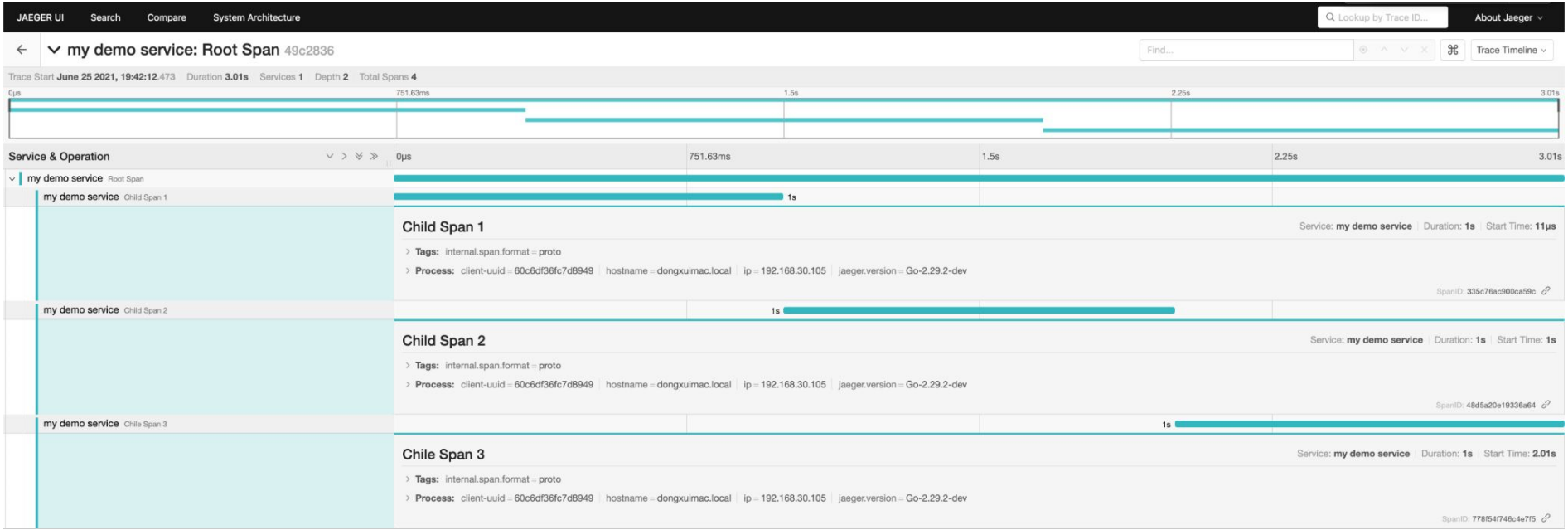


Example:

```go
func main() {
    closer := initTracer()
    defer closer.Close()

    (func() {
        span := opentracing.GlobalTracer().StartSpan("Root Span")
        defer span.Finish()

        (func() {
            span := span.Tracer().StartSpan("Child Span 1", opentracing.ChildOf(span.Context()))
            defer span.Finish()
            time.Sleep(time.Second)
        })()

        (func() {
            span := span.Tracer().StartSpan("Child Span 2", opentracing.ChildOf(span.Context()))
            defer span.Finish()
            time.Sleep(time.Second)
        })()

        (func() {
            span := span.Tracer().StartSpan("Chile Span 3", opentracing.ChildOf(span.Context()))
            defer span.Finish()
            time.Sleep(time.Second)
        })()
    })()
}
```

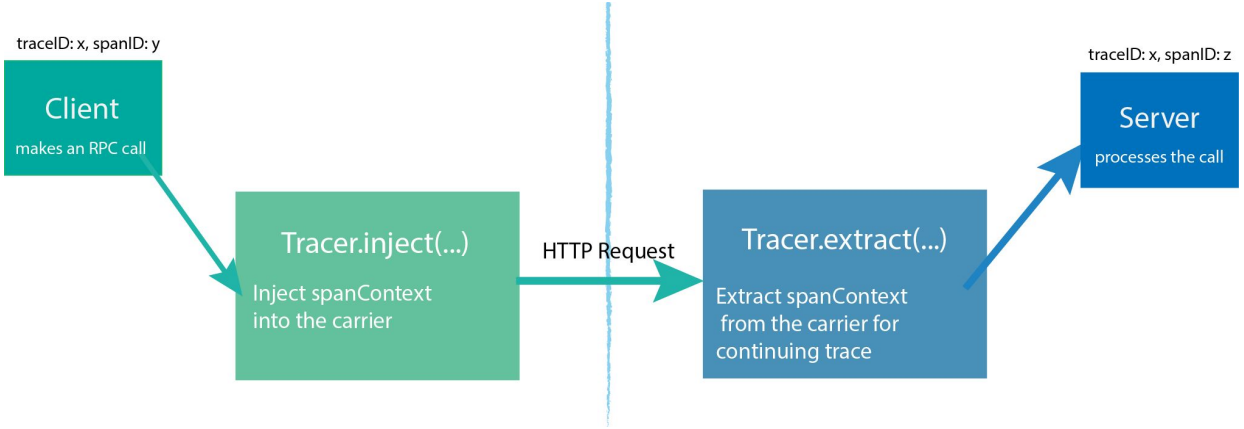
```go
func initTracer() io.Closer {
    cfg := jaegercfg.Configuration{
        ServiceName: "my demo service",
        Sampler: &jaegercfg.SamplerConfig{
            Type:  jaeger.SamplerTypeConst,
            Param: 1,
        },
        Reporter: &jaegercfg.ReporterConfig{},
    }

    tracer, closer, err := cfg.NewTracer()
    if err != nil {
        log.Fatal(err)
    }
    opentracing.SetGlobalTracer(tracer)

    return closer
}
```

Example:


JAEGER UI
Search
Compare
System Architecture
Lookup by Trace ID...
About Jaeger
my demo service: Root Span 49c2836
Find...
Trace Timeline
Trace Start June 25 2021, 19:42:12.473 Duration 3.01s Services 1 Depth 2 Total Spans 4
0µs
751.63ms
1.5s
2.25s
3.01s
Service & Operation
my demo service Root Span
my demo service Child Span 1
1s
Child Span 1
Service: my demo service | Duration: 1s | Start Time: 11µs
Tags: internal.span.format = proto
Process: client-uuid = 60c6df36fc7d8949 | hostname = dongxuimac.local | ip = 192.168.30.105 | jaeger.version = Go-2.29.2-dev
SpanID: 335c76ac900ca59c
my demo service Child Span 2
1s
Child Span 2
Service: my demo service | Duration: 1s | Start Time: 1s
Tags: internal.span.format = proto
Process: client-uuid = 60c6df36fc7d8949 | hostname = dongxuimac.local | ip = 192.168.30.105 | jaeger.version = Go-2.29.2-dev
SpanID: 48d5a20e19336a64
my demo service Chile Span 3
1s
Chile Span 3
Service: my demo service | Duration: 1s | Start Time: 2.01s
Tags: internal.span.format = proto
Process: client-uuid = 60c6df36fc7d8949 | hostname = dongxuimac.local | ip = 192.168.30.105 | jaeger.version = Go-2.29.2-dev
SpanID: 778f54f746c4e7f5

Carrier

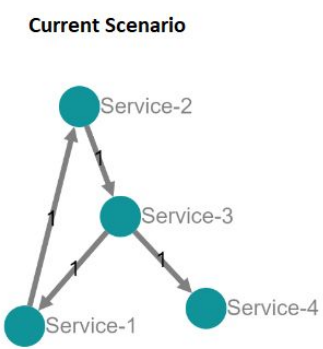
- Tracing 信息携带者
  - 解决跨服务调用的 Tracing metadata 序列化和反序列化抽象
- Inject / Extract
  - HTTPHeader
  - TextMap

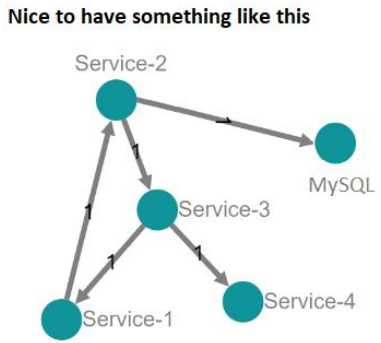


Example:
通过 HTTPHeader 作为 Carrier 来携带 Tracing 信息

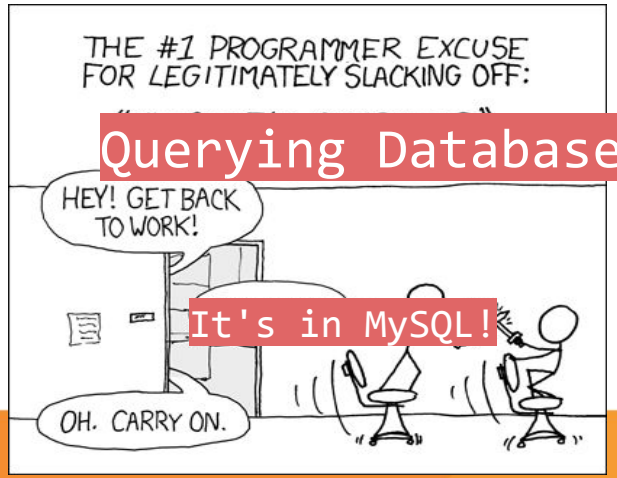

TOO EASY
WHAT'S NEXT?
memegenerator.net

如果我想回答一个问题


Current Scenario
Service-2
Service-3
Service-1
Service-4
1
1
1
1

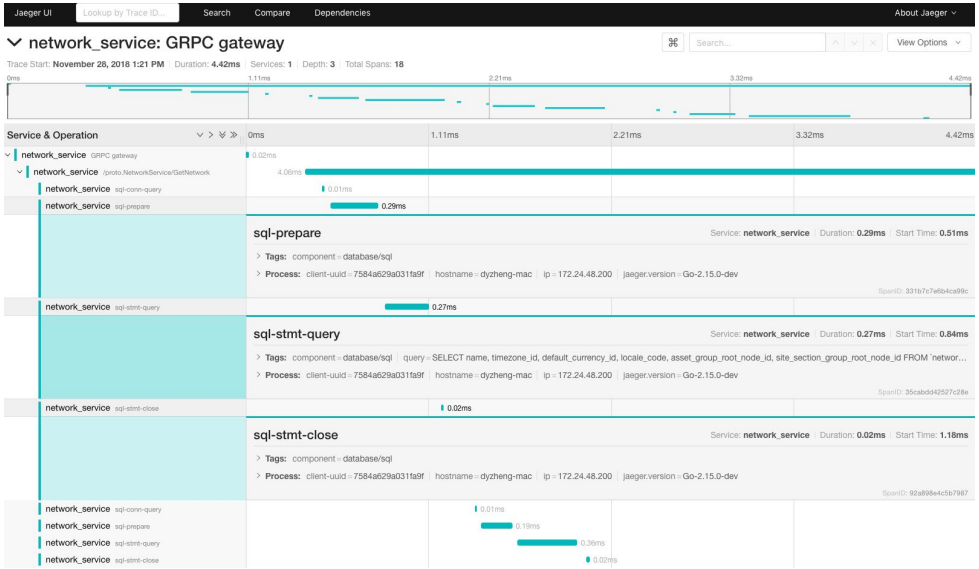

Nice to have something like this
Service-2
MySQL
Service-3
Service-1
Service-4
1
1
1
1


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
Querying Database!
HEY! GET BACK TO WORK!
It's in MySQL!
OH. CARRY ON.

instrumentedsql

luna-duclos/instrumentedsql: A sql driver that will wrap any other driver and log/trace all its calls




```
import (
    ...
    "github.com/go-sql-driver/mysql"
    "github.com/luna-duclos/instrumentedsql"
    "github.com/luna-duclos/instrumentedsql/opentracing"
    ...
)

    sql.Register("instrumented-mysql",
        instrumentedsql.WrapDriver(mysql.MySQLDriver{},
            instrumentedsql.WithTracer(opentracing.NewTracer(false)),
            instrumentedsql.WithOmitArgs(),
        ),
    )
    db, err := sql.Open("instrumented-mysql", dsn)
    // db, err := sql.Open("mysql", dsn)
```

instrumentedsql 原理

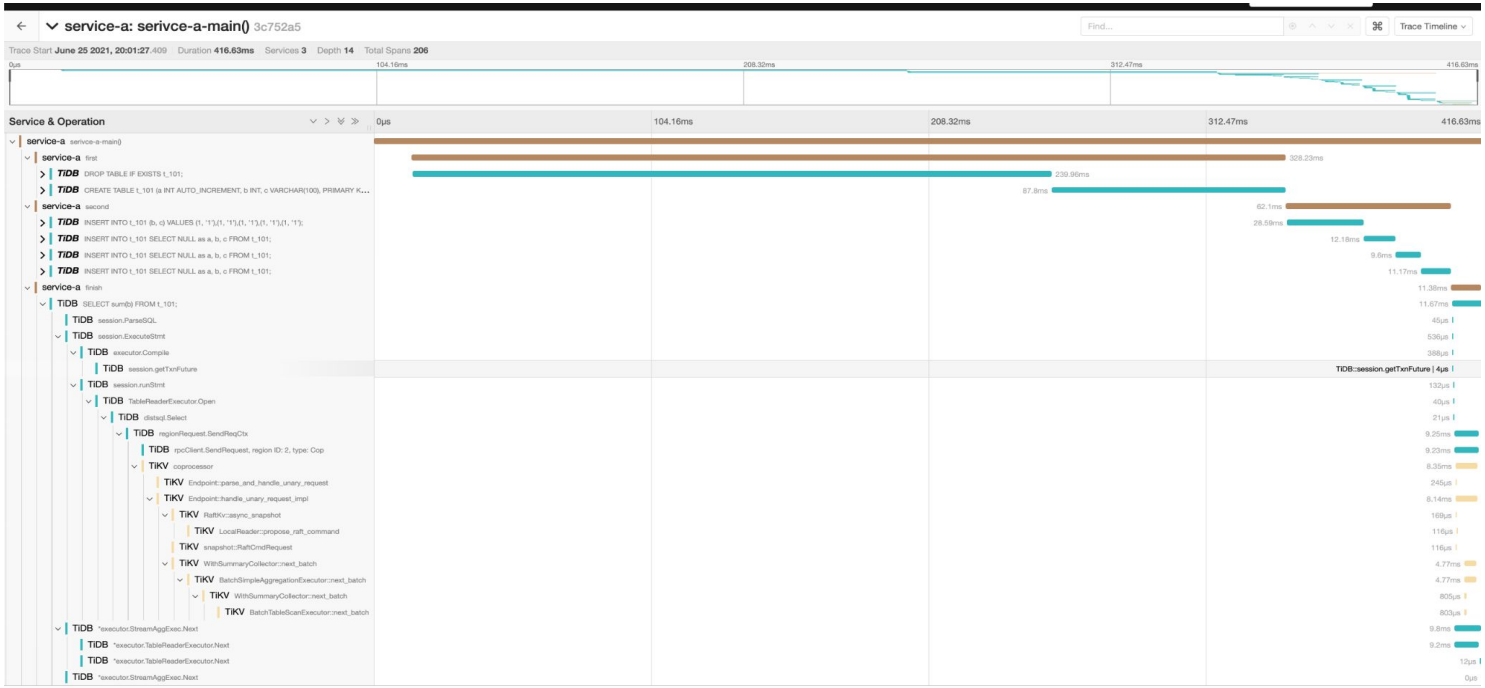
```
func (s wrappedStmt) Exec(args []driver.Value) (res driver.Result, err error) {
    if !s.hasOpExcluded(OpSQLStmtExec) {
            span := s.GetSpan(s.ctx).NewChild(OpSQLStmtExec)
            span.SetLabel("component", "database/sql")
            span.SetLabel("query", s.query)
            if !s.OmitArgs {
                    span.SetLabel("args", formatArgs(args))
            }
            start := time.Now()
            defer func() {
                    span.SetError(err)
                    span.Finish()
                    logQuery(s.ctx, s.opts, OpSQLStmtExec, s.query, err, args, start)
            }()
    }

    res, err = s.parent.Exec(args)
    if err != nil {
            return nil, err
    }

    return wrappedResult{opts: s.opts, ctx: s.ctx, parent: res}, nil
}
```

在 Driver 相关的调用加 Wrapper 插入 Span

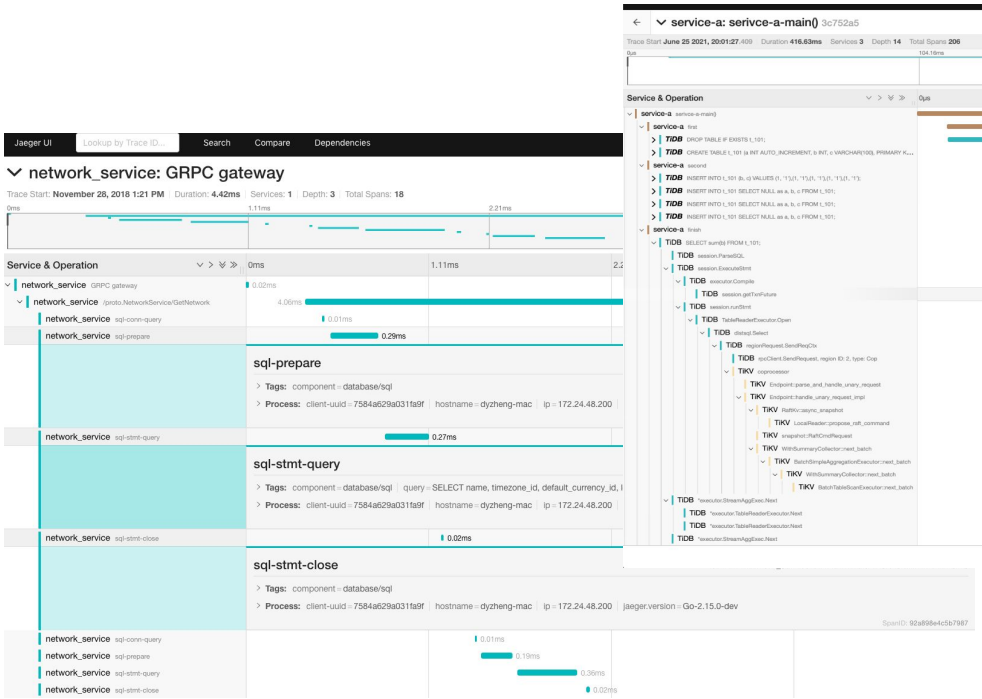
真实的数据库调用对我们来说还是黑洞

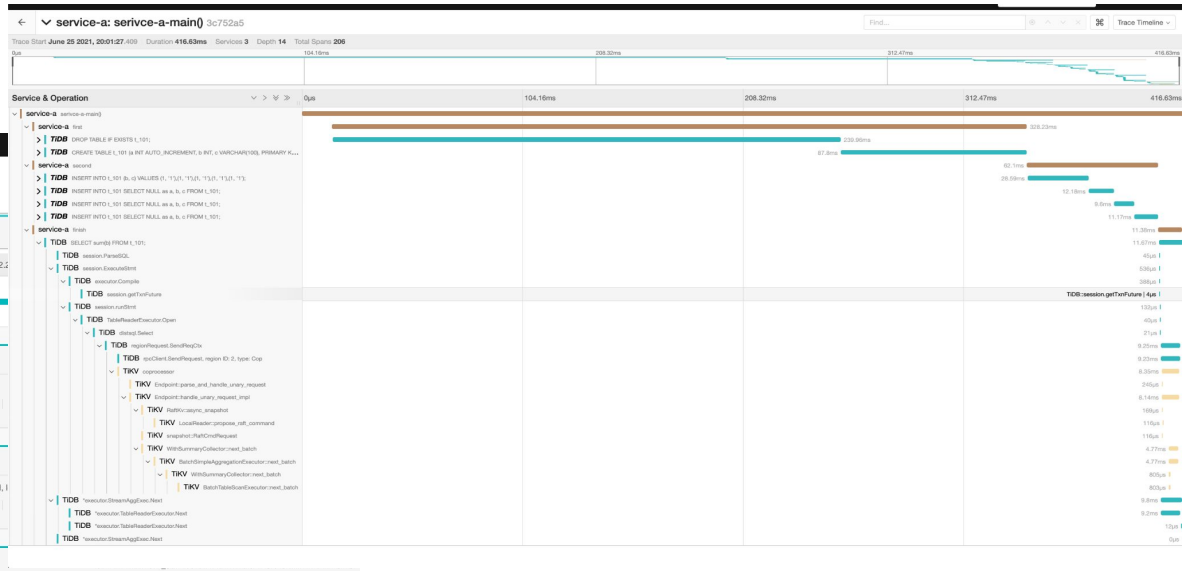

boredpanda.com

少废话先看东西


service-a: serivce-a-main() 3c752a5
Trace Start June 25 2021, 20:01:27.409 Duration 416.63ms Services 3 Depth 14 Total Spans 206
Service & Operation
0µs
104.16ms
208.32ms
312.47ms
416.63ms
service-a serivce-a-main()
service-a first
TiDB DROP TABLE IF EXISTS t_101;
TiDB CREATE TABLE t_101 (a INT AUTO_INCREMENT, b INT, c VARCHAR(100), PRIMARY K...
service-a second
TiDB INSERT INTO t_101 (b, c) VALUES (1, '1'),(1, '1'),(1, '1'),(1, '1'),(1, '1');
TiDB INSERT INTO t_101 SELECT NULL as a, b, c FROM t_101;
TiDB INSERT INTO t_101 SELECT NULL as a, b, c FROM t_101;
TiDB INSERT INTO t_101 SELECT NULL as a, b, c FROM t_101;
service-a finish
TiDB SELECT sum(b) FROM t_101;
TiDB session.ParseSQL
TiDB session.ExecuteStmt
TiDB executor.Compile
TiDB session.getTxnFuture
TiDB session.runStmt
TiDB TableReaderExecutor.Open
TiDB distsql.Select
TiDB regionRequest.SendReqCtx
TiDB rpcClient.SendRequest, region ID: 2, type: Cop
TiKV coprocessor
TiKV Endpoint::parse_and_handle_unary_request
TiKV Endpoint::handle_unary_request_impl
TiKV RaftKv::async_snapshot
TiKV LocalReader::propose_raft_command
TiKV snapshot::RaftCmdRequest
TiKV WithSummaryCollector::next_batch
TiKV BatchSimpleAggregationExecutor::next_batch
TiKV WithSummaryCollector::next_batch
TiKV BatchTableScanExecutor::next_batch
TiDB *executor.StreamAggExec.Next
TiDB *executor.TableReaderExecutor.Next
TiDB *executor.TableReaderExecutor.Next
TiDB *executor.StreamAggExec.Next
328.23ms
239.96ms
87.8ms
62.1ms
26.59ms
12.18ms
9.8ms
11.17ms
11.38ms
11.67ms
45µs
536µs
388µs
TiDB::session.getTxnFuture | 4µs
132µs
40µs
21µs
9.25ms
9.23ms
8.35ms
245µs
8.14ms
168µs
116µs
116µs
4.77ms
4.77ms
805µs
803µs
9.8ms
9.2ms
12µs
0µs

少废话先看东西


Jaeger UI
Lookup by Trace ID...
Search
Compare
Dependencies
network_service: GRPC gateway
Trace Start: November 28, 2018 1:21 PM | Duration: 4.42ms | Services: 1 | Depth: 3 | Total Spans: 18
Service & Operation
network_service GRPC gateway
network_service /proto.NetworkService/GetNetwork
network_service sql-conn-query
network_service sql-prepare
network_service sql-stmt-query
network_service sql-stmt-close
sql-prepare
Tags: component = database/sql
Process: client-uuid = 7584a629a031fa9f | hostname = dyzheng-mac | ip = 172.24.48.200
sql-stmt-query
Tags: component = database/sql | query = SELECT name, timezone_id, default_currency_id,
Process: client-uuid = 7584a629a031fa9f | hostname = dyzheng-mac | ip = 172.24.48.200
sql-stmt-close
Tags: component = database/sql
Process: client-uuid = 7584a629a031fa9f | hostname = dyzheng-mac | ip = 172.24.48.200 | jaeger.version = Go-2.15.0-dev


service-a: serivce-a-main() 3c752a5
Trace Start June 25 2021, 20:01:27.409 Duration 416.63ms Services 3 Depth 14 Total Spans 206
Service & Operation
0µs
104.16ms
208.32ms
312.47ms
416.63ms

Under the hood

- 核心思想：将 Tracer ID 和 Span Context 传递到 TiDB 内核中
- 具体做法：

1. 添加一个 Session Varible, 语法如： SET @@tidb_tracer_id=xxxxxx;
2. 在 Application 里将 Tracer ID 和 Span Context 序列化成字符串后传递给这个 Session Varible
3. 在 TiDB 体系内将 Tracer 信息反序列化后生成新的 Context
4. TiDB 和 TiKV 之间的通信是通过 gRPC, jaeger 对 gRPC 有着良好的支持

Tips: Jaeger client 提供了序列化和反序列化的 实现：

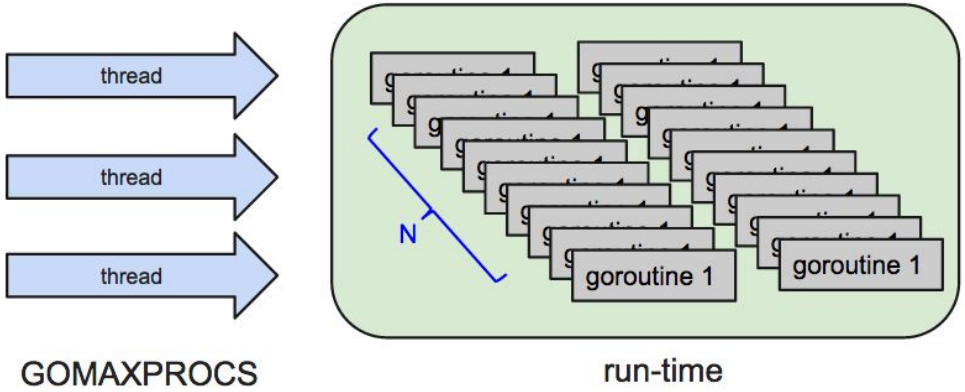
```
func (c SpanContext) String() string
func ContextFromString(value string) (SpanContext, error)
```

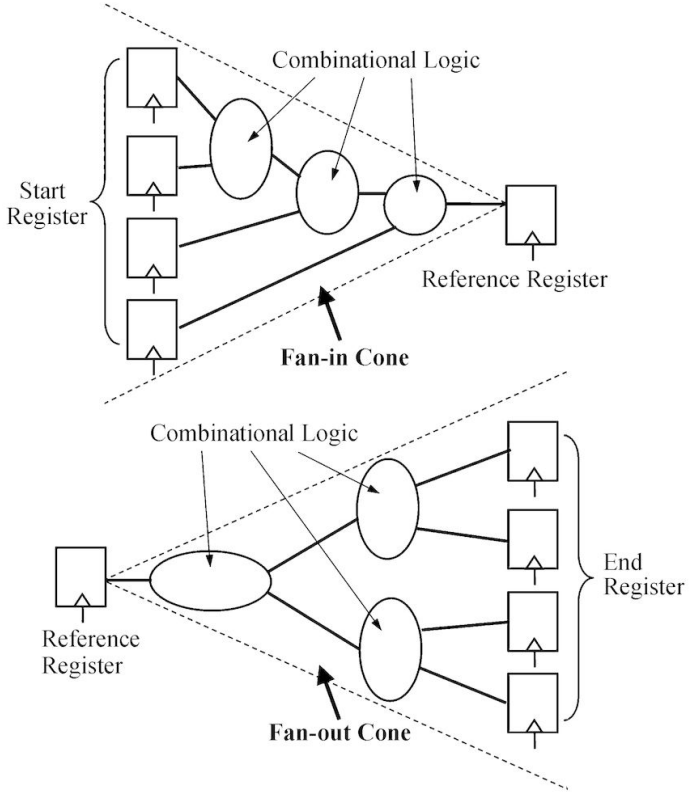

TOO EASY
WHAT'S NEXT?
memegenerator.net

让我们更近一步？

「我确实能看到的时间花在哪里了，但是为什么花了那么长时间？」

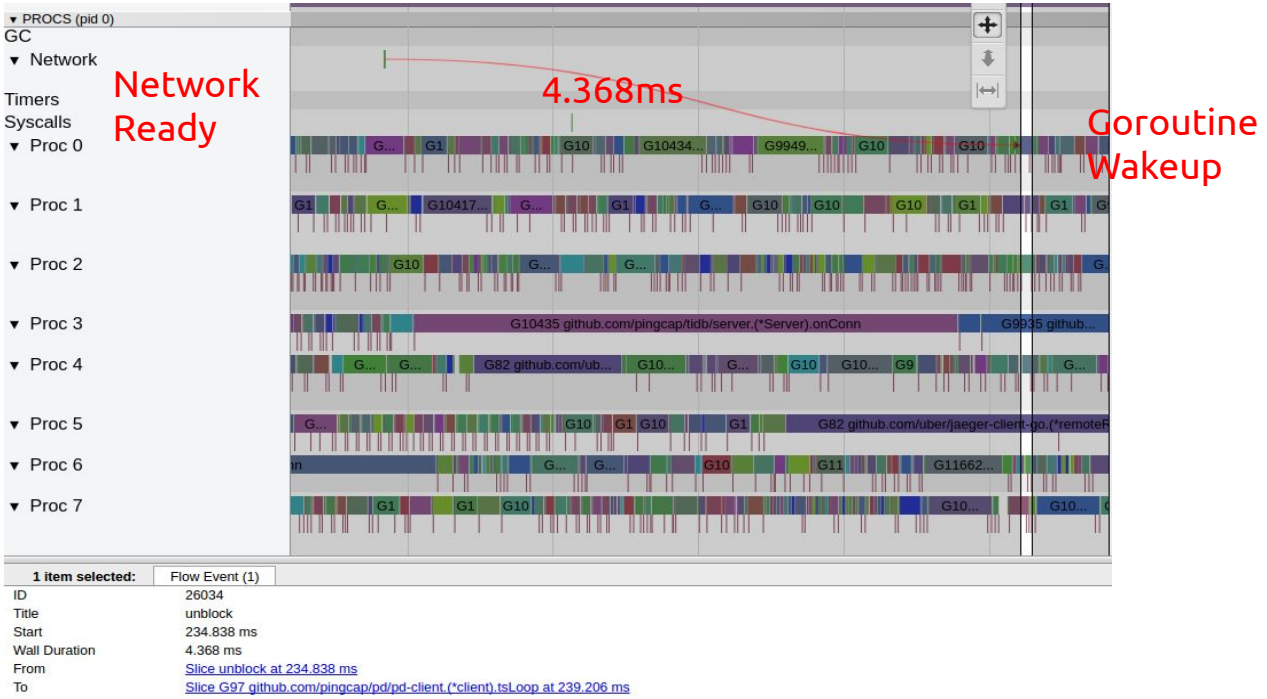
总所周知





灵魂拷问:「到底是我的 SQL 写得💩，还是 Goroutine 调度不周？」

go tool trace



go tool trace

- 优点：好用，好看（UI）
- 缺点：性能损耗太大，不能一直开着

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)

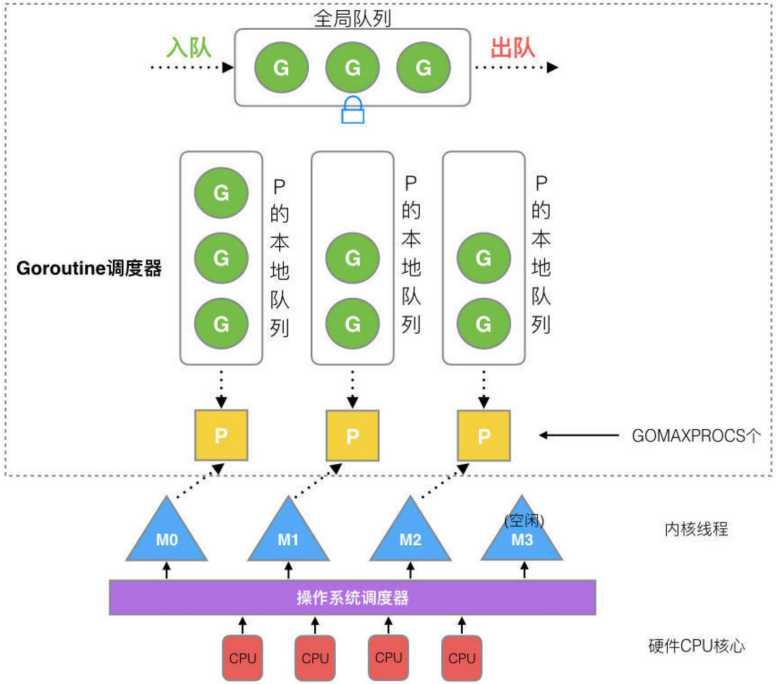
Goroutine	Total		Execution	Network wait	Sync block	Blocking syscall	Scheduler wait	GC sweeping	GC pause
15887	1000ms		995ms	2818μs	0ns	41μs	1341μs	3113μs (0.3%)	1760μs (0.2%)

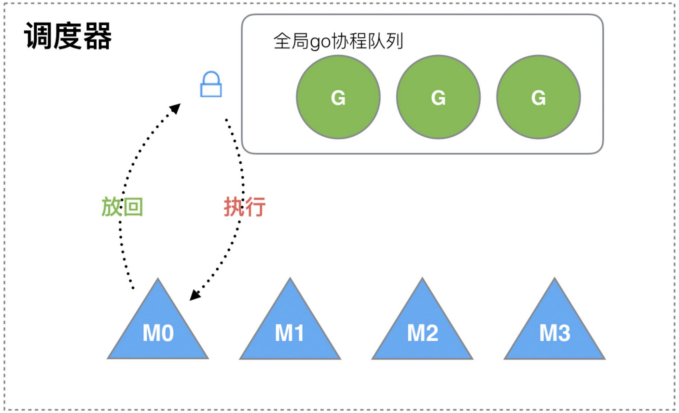
Trace in Go runtime

- go tool trace 的原理是？

Trace 会 Go Runtime 的代码中打桩收集 CPU time，在 Goroutine 开始执行时记录 start_run_time, 在调度退出执行时记录 end_run_time，累加 (end_run_time - start_run_time) 即为这个 goroutine 的 CPU time。

A little bit about Go runtime





https://learnku.com/articles/41728

```go
package runtime

import (
	"runtime/internal/atomic"
	"runtime/internal/sys"
	"unsafe"
)

// Event types in the trace, args are given in square brackets.
const (
	traceEvNone              = 0  // unused
	traceEvBatch             = 1  // start of per-P batch of events [pid, timestamp]
	traceEvFrequency         = 2  // contains tracer timer frequency [frequency (ticks per second)]
	traceEvStack             = 3  // stack [stack id, number of PCs, array of {PC, func string ID, file string ID, line
	traceEvGomaxprocs        = 4  // current value of GOMAXPROCS [timestamp, GOMAXPROCS, stack id]
	traceEvProcStart         = 5  // start of P [timestamp, thread id]
	traceEvProcStop          = 6  // stop of P [timestamp]
	traceEvGCStart           = 7  // GC start [timestamp, seq, stack id]
	traceEvGCDone            = 8  // GC done [timestamp]
	traceEvGCSTWStart        = 9  // GC STW start [timestamp, kind]
	traceEvGCSTWDone         = 10 // GC STW done [timestamp]
	traceEvGCSweepStart      = 11 // GC sweep start [timestamp, stack id]
	traceEvGCSweepDone       = 12 // GC sweep done [timestamp, swept, reclaimed]
	traceEvGoCreate          = 13 // goroutine creation [timestamp, new goroutine id, new stack id, stack id]
	traceEvGoStart           = 14 // goroutine starts running [timestamp, goroutine id, seq]
	traceEvGoEnd             = 15 // goroutine ends [timestamp]
	traceEvGoStop            = 16 // goroutine stops (like in select{}) [timestamp, stack]
	traceEvGoSched           = 17 // goroutine calls Gosched [timestamp, stack]
	traceEvGoPreempt         = 18 // goroutine is preempted [timestamp, stack]
	traceEvGoSleep           = 19 // goroutine calls Sleep [timestamp, stack]
	traceEvGoBlock           = 20 // goroutine blocks [timestamp, stack]
	traceEvGoUnblock         = 21 // goroutine is unblocked [timestamp, goroutine id, seq, stack]
	traceEvGoBlockSend       = 22 // goroutine blocks on chan send [timestamp, stack]
	traceEvGoBlockRecv       = 23 // goroutine blocks on chan recv [timestamp, stack]
	traceEvGoBlockSelect     = 24 // goroutine blocks on select [timestamp, stack]
	traceEvGoBlockSync       = 25 // goroutine blocks on Mutex/RWMutex [timestamp, stack]
	traceEvGoBlockCond       = 26 // goroutine blocks on Cond [timestamp, stack]
	traceEvGoBlockNet        = 27 // goroutine blocks on network [timestamp, stack]
```

hack runtime 的思路：

follow the tracing event.

https://github.com/golang/go/blob/master/src/runtime/trace.go

Tracing Runtime 伪代码

```
type g struct {
    goid int64 // goroutine id.
    . . .

    statCtx *t // g 所在的 task_group，默认创建 g 时从
parent g 继承。
    // lastSchedTime is the nanotime of the last time
a goroutine was scheduled into a P.
    lastSchedTime int64
}

type t struct {
    schedtick uint64 // increment atomically on every
scheduler call
    nanos     uint64 // cumulative slices of CPU time
used by the task group, in nanoseconds
}
```

```
// Goroutine 开始运行时，记录开始信息
gp.lastSchedTime = nanotime()
if gp.statCtx != nil {
        atomic.Xadd64(&gp.statCtx.schedtick, 1)
}

…

// Goroutine 暂停运行时，收集执行时长
endTickTime := nanotime()
lastTime := _g_.m.curg.lastSchedTime
atomic.Xadd64(&_g_.m.curg.statCtx.nanos,
endTickTime-lastTime)
```

PingCAP 的一个实验性 Go rumtime 分支

https://github.com/crazycs520/go/tree/stats-dev2

Tracing runtime

缺点:「你得自己维护一个 Go 分支」

有没有别的办法？

Tracing runtime

有

Profiler Label

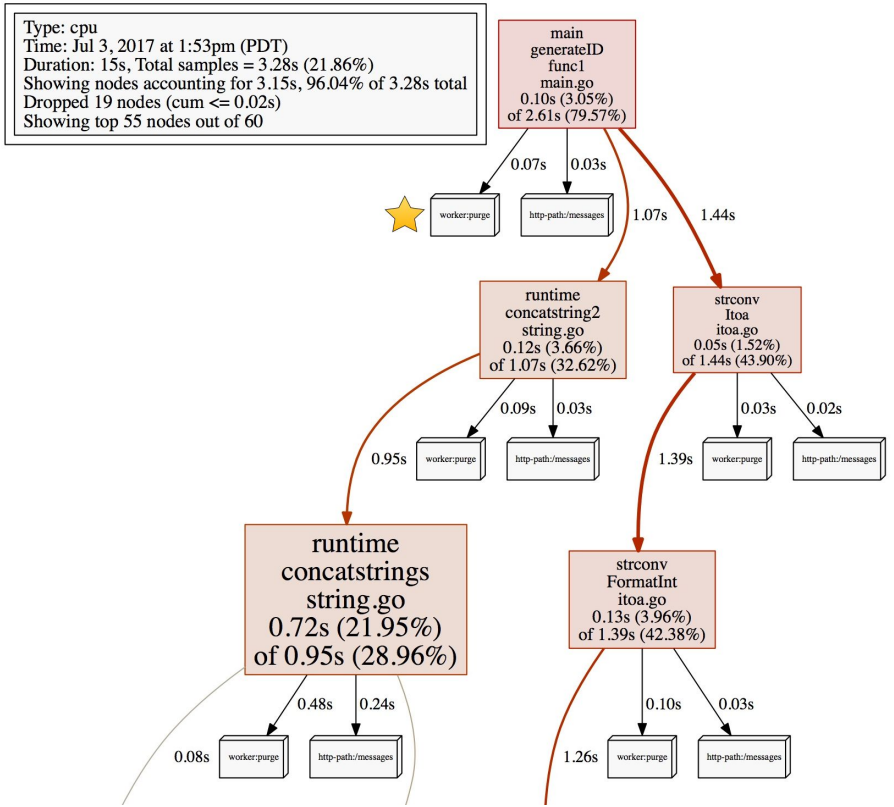
喜闻乐见的 pprof 变得更强大了

Go 1.9 is introducing **profiler labels**, a way to add **arbitrary key-values** to the samples collected by the CPU profiler. CPU profilers collect and output hot spots where the CPU spent most time in when executing. A typical CPU profiler output is primarily reports the location of these spots as function name, source file/line, etc. By looking at the data, you can also examine which parts of the code invoked these spots. You can also filter by invokers to have more granular understanding of certain execution paths.

https://github.com/golang/proposal/blob/master/design/17280-profile-labels.md

Profiler Label

```
labels := pprof.Labels("worker", "purge")
pprof.Do(ctx, labels, func(ctx context.Context)
{
    // Do some work...

    go update(ctx) // propagates labels in ctx.
})
```



使用 Profiler Label 来统计 SQL 执行的 CPU 时间

思路：

- 在 SQL 解析器完成解析后，生成 SQL 的指纹信息（SQL Digest）
- 生成带着这个 Digest 的 Context
- 使用 pprof.SetGoroutineLables 对当前

```
// After parse and calculated SQL digest, set the sql digest label
ctx = pprof.WithLabels(ctx, pprof.Labels("sql", sql_digest))
pprof.SetGoroutineLabels(ctx)
```

func SetGoroutineLabels 1.9 Source

```
func SetGoroutineLabels(ctx context.Context)
```

SetGoroutineLabels sets the current goroutine's labels to match ctx. A new goroutine inherits the labels of the goroutine that created it. This is a lower-level API than Do, which should be used instead when possible.

pprof 的性能损耗怎么办？

问：开 pprof 对性能损耗太大

答：走得远了，不要忘了我们为什么出发，我们不是做 profile，tracing 可以采样

```
func Run() {
	go startCPUProfileWorker()
	go startAnalyzeProfileWorker()  // 另一个 goroutine 异步分析 profile 结果
}

func startCPUProfileWorker() {
	for {
		buf := getBuffer()
		if err := pprof.StartCPUProfile(buf); err != nil {
			return
		}
		// 就 profile 一秒
		sp.sleep(time.Second * 1)
		pprof.StopCPUProfile()
		taskCh <- buf // 将结果 profile 结果异步提交给 AnalyzeProfileWorker
	}
}
```

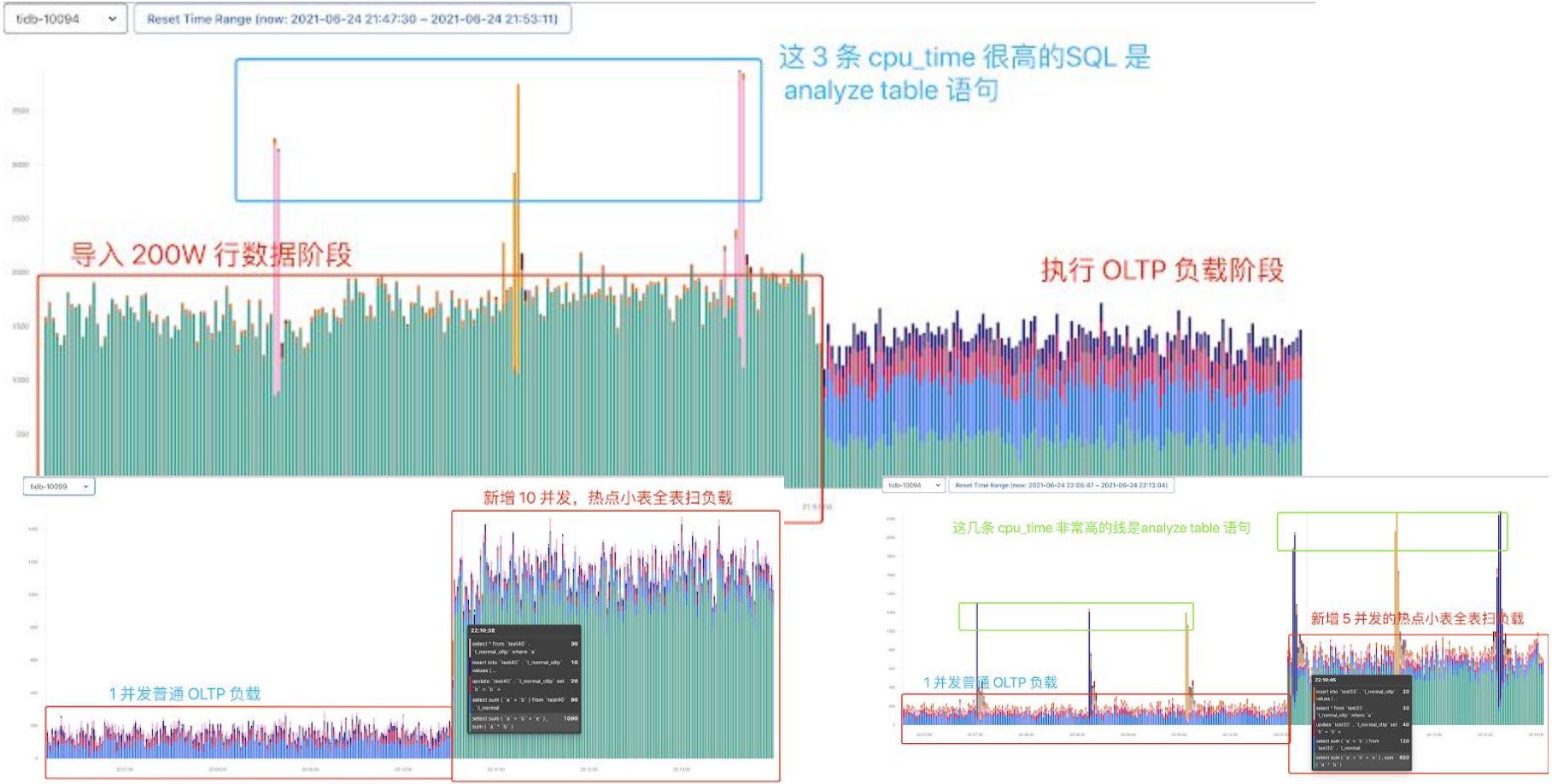
Result

测试负载：sysbench `oltp_point_select`
并发：48（TiDB tidb avg cpu usage 50%）
Go 版本: go 1.16

	avg QPS	损耗	avg P99 Duration	损耗	avg P50 Duration	损耗
Master	102.1k	\	1.83ms	\	0.301ms	\
profile + labels	98.9k	3.1%	1.87ms	2.1%	0.307ms	1.9%
Runtime Hacking	98.9k	3.1	1.87ms	2.1%	0.304ms	0.98%

sysbench `oltp_read_write`

	avg QPS	损耗	avg P99 Duration	损耗
Master	63.2k	\	22.4ms	\
profile + labels	62.9k	0.4%	23.1ms	3%
Runtime Hacking	62.8k	0.6%	22.6ms	0.8%


tidb-10094
Reset Time Range (now: 2021-06-24 21:47:30 ~ 2021-06-24 21:53:11)
这 3 条 cpu_time 很高的SQL 是 analyze table 语句
导入 200W 行数据阶段
执行 OLTP 负载阶段
tidb-10099
新增 10 并发，热点小表全表扫负载
1 并发普通 OLTP 负载
tidb-10094
Reset Time Range (now: 2021-06-24 22:08:47 ~ 2021-06-24 22:13:04)
这几条 cpu_time 非常高的线是analyze table 语句
1 并发普通 OLTP 负载
新增 5 并发的热点小表全表扫负载

TiDB

Thanks

h@pingcap.com

公众号: PingCAP