GIAC | BEIJING Dec.12.16-17

架構 ARCHNOTES 高可用架构

Real world Rust

Why and how we use Rust in TiKV

黄东旭 PingCAP

技术架构未来

thegiac.com

关于我

- 黄东旭 Dongxu_Huang
- Open source hacker / Infrastructure Engineer
- MSRA / Netease / Wandoujia / PingCAP
- CTO of PingCAP
- Codis / TiDB / TiKV
- Go / Python / Rust ...

What's Rust

- Yet another system programming language
  - Maintain by Mozilla
- Stable: 1.13
  - Stable enough for production use
- Alternative to C/C++

Why Rust?

- Safe
  - Thread safety guarantee
  - Segfaults free
- Blazingly fast
  - llvm backend
  - No GC
  - No internal run-time
- Modern tool chain
  - Modern package management tools
  - Embedded unit test framework
- Stdlibs

Rust 哲学

- Zero-cost abstraction
- 内存安全
  - Build-in RAII / Ownership
  - No NULL ptr
- 万物皆有所有权
  - Ownership system
  - Mutable and immutable reference
- 线程安全
- 不相信程序员

内存安全

- 所有权
- RAII / Lifetime

内存安全

```
fn do_vec(v: Vec<u32>) {}

fn main() {
    let a = vec![1,2,3];
    do_vec(a);
    println!("{}", a)
}
```

内存安全

```
error[E0382]: use of moved value: `a`
 --> src/main.rs:6:20
  |
5 |     do_vec(a);
  |            - value moved here
6 |     println!("{}", a[0])
  |                    ^ value used here after move
```

内存安全

```rust
fn main() {
    let mut x = vec![1,2,3];
    let y = &mut a;
    let z = &a;
    ...
}
```

内存安全

```
error[E0502]: cannot borrow `a` as immutable because it
is also borrowed as mutable
  --> src/main.rs:8:15
   |
7  |     let y = &mut a;
   |                  - mutable borrow occurs here
8  |     let z = & a;
   |               ^ immutable borrow occurs here
```

内存安全

```
fn main() {
    let mut x = vec![1,2,3];
    let y = &mut a;
    let z = &mut a;
    ...
}
```

内存安全

```
error[E0499]: cannot borrow `a` as mutable more than
once at a time
  --> src/main.rs:8:18
   |
7  |     let y = &mut a;
   |             - first mutable borrow occurs here
8  |     let z = &mut a;
   |             ^ second mutable borrow occurs
here
```

这像什么？

- 一个变量，可以同时进行多个 immutable 的 borrow，但只允许一个 mutable 的 borrow
- 这个其实跟 read-write lock 很相似，同时允许多个读锁，但一次只允许一个写锁

Lifetime

```
struct A<'a> {
    b: &'a u32
}
fn main() {
    let b = 10;
    let mut a = A{ b: &b };
    {
        let c = 10;
        a.b = &c;
    }
    println!("{}", a.b);
}
```

Lifetime

```
error: `c` does not live long enough
  --> src/main.rs:11:16
   |
11 |            a.b = &c;
   |                   ^
   |
note: reference must be valid for the block suffix
following statement 1 at 8:27...
  --> src/main.rs:8:28
   |
8  |     let mut a = A{ b: &b };
```

```go
var n = 1
go func() {
    n += 1
}()
go func() {
    n += 1
}()
```

Send / Sync 原语

- 如果 T: Send，那么可以安全的在线程间**传递** T
  - 不同线程即使销毁也无所谓
- 如果 T: Sync，那么可以安全的在线程间**共享** T
- Rust 的类型推导系统和编译检查跨线程传递和共享的对象是否满足 Send + Sync

TiKV

- 大规模分布式 Key-Value 数据库
- 支持 ACID 跨行事务支持
- 支持 MVCC 无锁的快照读
- 构建于 Raft 之上，不依赖分布式文件系统
  - 更少的第三方依赖
  - 更高的性能（低延迟）
- 配合 TiDB 使用，需要有健全的逻辑实现 SQL 层的下推算子

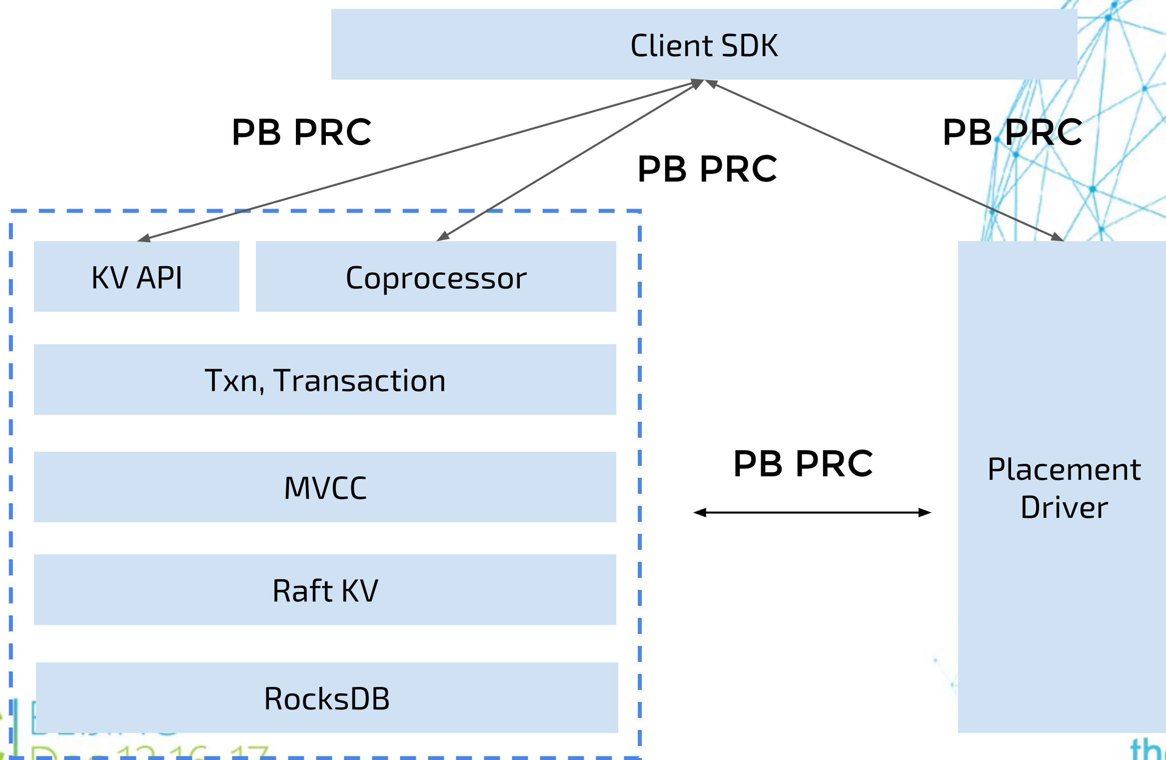
TiKV 的前置需求

- 极高的性能要求，尽可能低延迟，而且延迟需要稳定
- 设计分布式系统的逻辑，极其复杂
  - Raft
  - Multi-Raft
  - 分布式测试框架
- 和 C 的模块大量交互
- 开发人员和时间不足
  - 5 人的团队，希望半年左右 发布第一个可用版本

TiKV 的语言选型参考

- Go
- C++11
- Rust

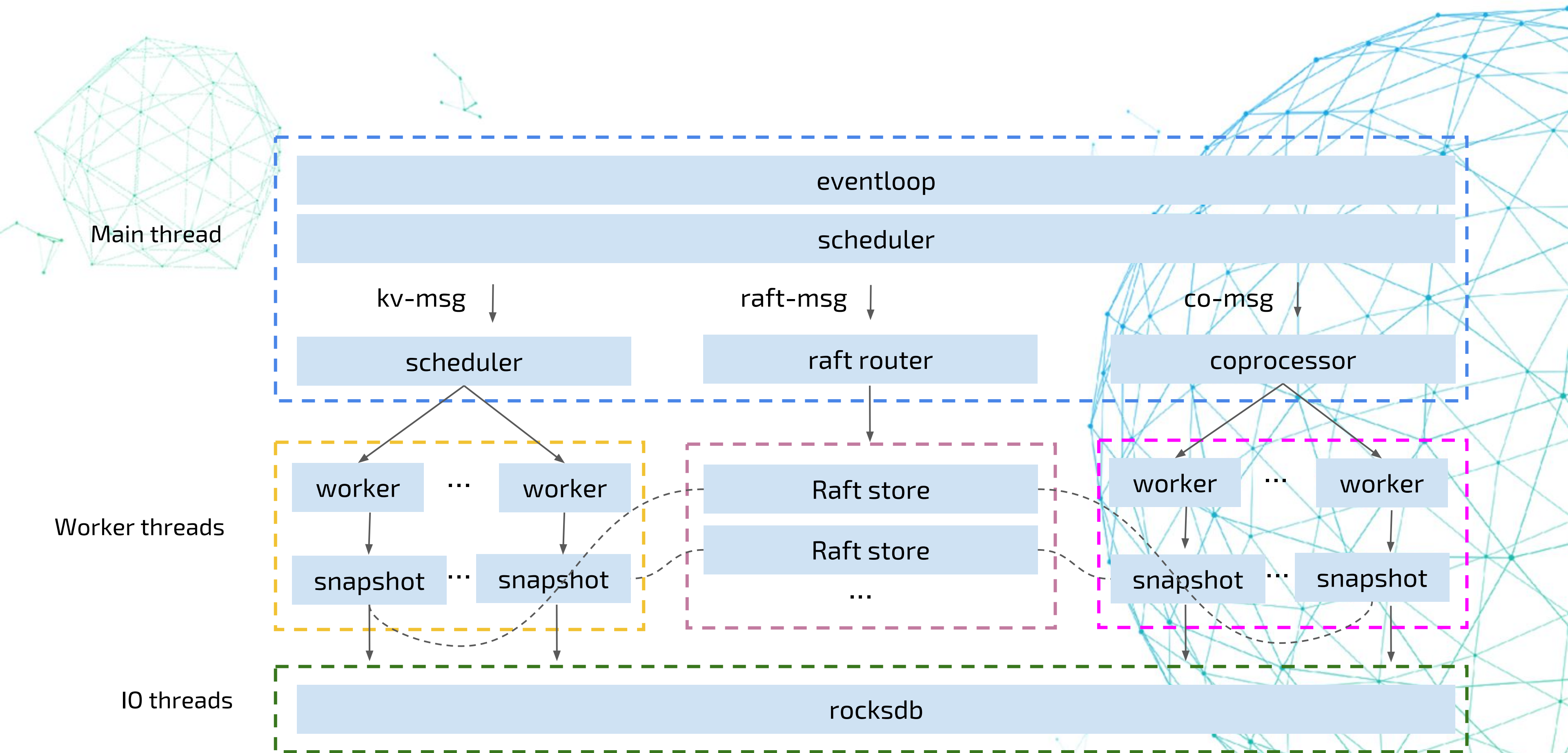
TiKV 整体架构



GIAC | Dec.12.16-17
thegiac.com

主要组件

- Asynchronous IO
  - mio
- Storage engine
  - RocksDB
- RPC
  - Protobuf PRC
- Metrics
  - Prometheus
- futures-rs
  - WIP
- rust-grpc
  - WIP


Main thread
eventloop
scheduler
kv-msg
raft-msg
co-msg
scheduler
raft router
coprocessor
Worker threads
worker
...
worker
Raft store
Raft store
...
worker
...
worker
snapshot
...
snapshot
snapshot
...
snapshot
IO threads
rocksdb

跨线程通信 - 如何选择

- Channel
  - 异构线程之间传递对象
- Arc + Mutex
  - 同构的工作线程间共享对象

Rust 和 C

- Rust 对 C 的调用没有任何 overhead
- C 区域的调用没法保证安全
- TiKV 场景的特殊性
  - RocksDB

周边工具

- cargo
- clippy
- rustfmt
- kcov
- perf + flamegraph

Rust 2017 Roadmap

- 优化学习曲线
- 完善工具链和 IDE
- 补全异步和并发编程工具包
- 有足够的工具和范式开发高性能、健壮的、可扩展的后端服务程序
- 大多数常用包都达到 1.0 的稳定性
- ...

总结

- Rust 是门好语言，可以简单类比成更现代的 C++
  - 更少的代码写出更安全的程序
  - 代价是早期更高的学 习成本
- 解决问题的架构和范式万变不离其宗
  - 并不是靠换个语言重写一遍就解决 问题
- TiKV 是一个性能极高且极其稳定的新一代分布式 kv 数据库 :)

谢谢

Q & A