高性能高可用的微服务框架TarsGo的腾讯实践

陈明杰
腾讯
jessemjchen@tencent.com

探探 Gopher China 2019

Tars介绍

- Tars是一个支持多语言、内嵌服务治理功能，与Devops能很好协同的微服务框架

OSS

Devops
- 代码管理
- 代码编译
- 自动化测试
- 持续部署
- 灰度发布

服务治理

服务注册/发现	负载均衡	熔断	服务配置
Metric监控	日志聚合	自定义监控	Set模型
分布式跟踪	自动区域感知	流量管理	过载保护

开发框架

RPC：同步、异步、单向

协议：Tars、TUP、SSL、PB、Http1/2、自定义协议

语言
- C++
- JAVA
- Nodejs
- PHP
- GO

基础设施（物理机、虚拟机、容器）

开发框架 – 基于TARS协议的RPC框架实现快速开发

```
module TestApp {
struct LoginInfo{
    0 require string sid;
    1 require string code;
};
struct ProfileInfo{
    0 require string nick;
    1 optional int level;
};
interface TestServant{
    int test(int qq,LoginInfo li,out ProfileInfo pi);
};
};
```

```
func (_obj *TestServant) Test(Qq int32, Li *LoginInfo, \
Pi *ProfileInfo, _opt ...map[string]string) (ret int32, err error) {

    var length int32
    var have bool
    var ty byte
    _os := codec.NewBuffer()
    err = _os.Write_int32(Qq, 1)
    if err != nil {
        return ret, err
    }

    err = Li.WriteBlock(_os, 2)
    if err != nil {
        return ret, err
    }
    ... ...
```

- ✓ 版本兼容性好
- ✓ 高性能
- ✓ 多语言支持便利

接口代码自动生成，实现业务快速开发

服务治理– 整体思路

熔断策略：
连续超时或失败次数超过阈值
一段时间内超时比率阈值

registry

getRoute(“xxxsvr”);

client

轮询、hash、权重

xxxsvr — keep alive — node

xxxsvr — keep alive — node

Metric 监控统计

Conf 配置中心

log 日志聚合

…

auth 认证鉴权

zipkin 分布式跟踪

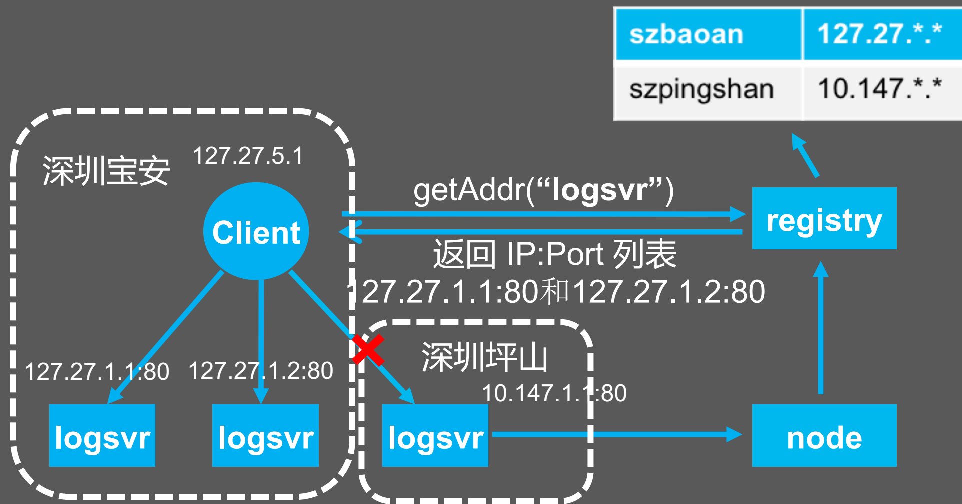
开发框架、Registry、node和基础服务集群协同工作，透明完成服务发现/注册、负载均衡、鉴权、分布式跟踪等服务治理相关工作

服务治理– 自动区域感知

◆常规的负载均衡方式面对跨地区或者跨机房部署的服务会因为网络原因造成延时增大

◆使用不同服务名来解决该问题时会带来繁重的运维工作

◆通过Registry和开发框架配合实现自动区域感知



自动区域感知优势

a) 运维简单

b) 降低延时减少带宽消耗

c) 更强的容灾能力

服务治理– 分布式跟踪

请求

Span:cs,cr

Gateway

服务器0

❶

❻

Span:sr,ss
Span:cs,cr
Span:cs,cr

Service1

服务器1

❷

❸

❹

❺

Span:sr,ss

Service2

服务器2

Span:sr,ss

Service3

服务器3

DB

跟踪数据落盘

Zipkin

界面展示

利用开源的Zipkin实现分布式跟踪

框架内部嵌入跟踪锚点使用对业务透明

OSS – 运营web化

提供Open API ，可定制自己的OSS系统

TARS | 服务管理 | 运维管理

- Prajna
- Robin
- StressTest
- Test
- TestApp
  - set
  - AccountServer
  - HelloJavaServer
  - HelloServer
  - OverloadServer
  - ProductServer
  - TestJavaClient
  - TracingJavaFiveServer
  - TracingJavaFourServer
  - TracingJavaOneServer
  - TracingJavaThreeServer
  - TracingJavaTwoServer
- TestJava
- TestTaf
- ddd
- denisApp
- tars
- windweitest

服务管理 | 发布管理 | 服务配置 | 服务监控 | 特性监控

配置列表

添加配置

	服务名称	文件名称	最后修改时间	操作
◉	TestApp.TracingJavaFourServer	application.properties	2018-05-14 21:34:24	修改配置 | 删除配置 | 查看内容 | 查看历史

引用文件列表

添加引用文件

服务名称	节点	文件名称	最后修改时间	操作
TestApp		TestApp.conf	2018-06-18 20:50:29	删除引用 | 查看引用内容 | 查看历史

节点配置列表

PUSH配置文件

☐	服务名称	节点	文件名称	最后修改时间	操作

TarsGo研发背景

统一：

微服务迭代交接节奏快，go的编程风格和设计统一，适合多人迭代开发

高效：

实际应用过程中除了大规模后台服务场景，更多的是运营小工具场景，使用go开发运营工具也更快

时代：

docker/k8s/etcd等开源项目大规模应用go语言，需要跟上云时代

学习成本：
- 关键字少：新人一周上手开发，C++转GO几乎无门槛
- 语言设计直观：新接手业务可以快速开发，没有奇怪的语法糖与复杂的定义

开发难度：
- 编译速度快：普通办工机即可编译
- 库提供源码：快速排查引用库问题
- 自带工具库：pprof、交叉编译、文档生成

版本管理：
- 远程git管理：go get安装与更新库，引用库集中管理

运营管理：
- 静态编译：运行程序不会对第三方库产生依赖
- 只依赖系统调用：不依赖glibc，对操作系统版本无限制，线上多环境运营至关重要

TarsGo架构

tars2go

- BNF词法分析
- BNF语法分析
- 代码生成
- Tars协议序列化 Pacakge codec

pacakge tars

- 特性监控上报 goroutine
- 远程日志上报 goroutine
- Stat监控上报 goroutine
- 路由刷新 goroutine

client

- Communictor
- Servantproxy
  - endpointmanager
    - adapter
    - adapter
- SendQueue chan
- Connection
  - net.Conn
  - net.Conn

server

- UDP/TCP/HTTP Listener
- Receive routine
  - connection
    - Net.Conn
    - Net.Conn
- RecvQueue chan
- Handler
  - worker goroutine
    - dispatcher
  - worker goroutine
    - dispatcher

熔断，SET支持，，PB协议支持，一致性hash调用
zipkin调用链，自定义插件，无损热重启… …

lib

- 一致性ash
- 消息通道
- logger
- 模板解析
- 远程配置
- 协程池
- … …

Tars编解码优化

问题：

Tars协议编解码情况性能低下，tars2go依赖cpp代码

解决方案：

尽可能减少数据copy，使用指针传递返回数据
类型去掉反射，使用tars协议type类型
纯go实现了AST，抛弃了yacc与bison，实现了框架从生成到实现的纯go化

```go
_i, _err = _is.Read(reflect.TypeOf(_obj.M_iVersion), 1, true)
if _err != nil {
    return _err
}
if _i != nil {
    _obj.M_iVersion = _i.(int16)
}
_i, _err = _is.Read(reflect.TypeOf(_obj.M_cPacketType), 2, true)
if _err != nil {
    return _err
}
```

```go
err = _is.Read_int16(&st.IVersion, 1, true)
if err != nil {
    return err
}

err = _is.Read_int8(&st.CPacketType, 2, true)
if err != nil {
    return err
}
```

Timer优化

问题：

见 https://go-review.googlesource.com/c/go/+/34784

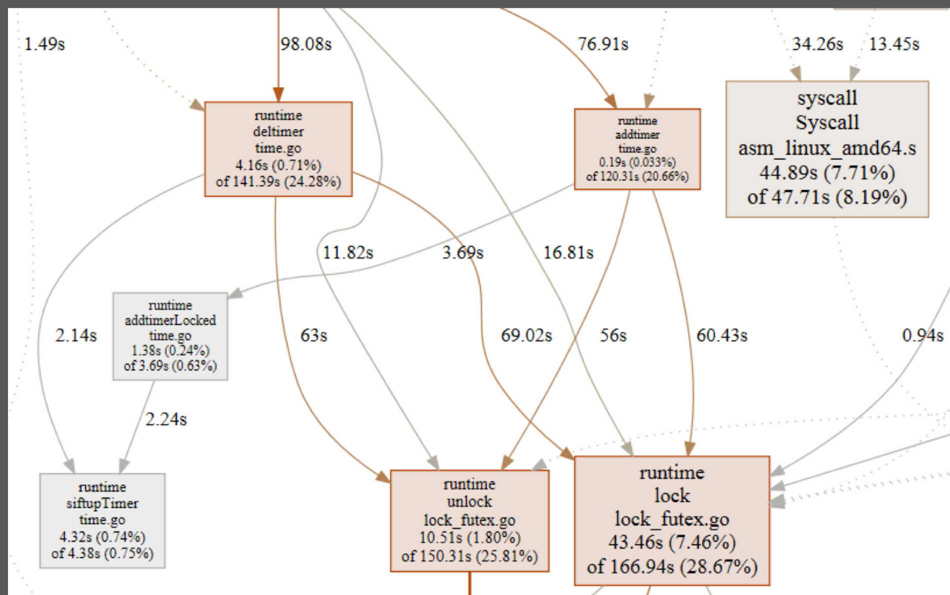
Muti-cpu场景下，timer性能低下

解决方案：

升级Go到最新的1.10.3及以上版本

Use per-P timers, so each P may work with its own timers

基于时间轮算法实现自己的timer



Net包的SetDeadline调用性能问题

问题：

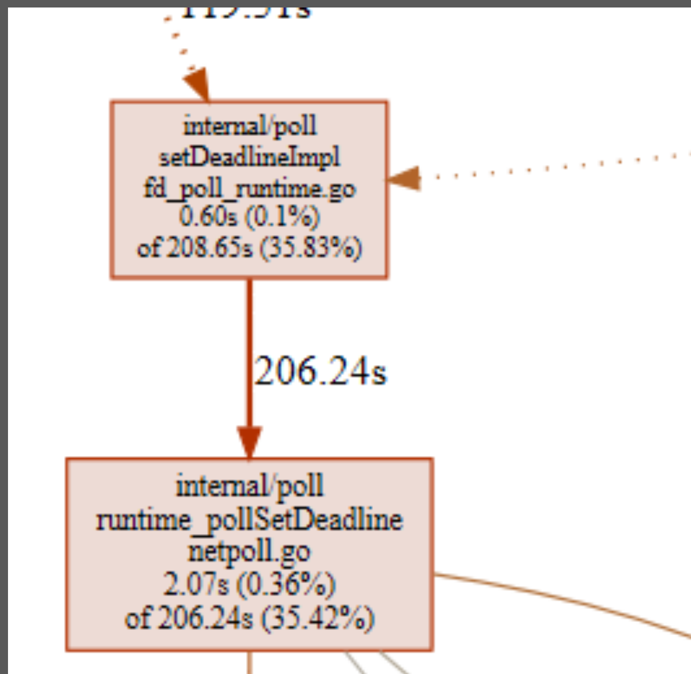
使用pacakge net 的SetDeadline /SetWriteDeadline/SetReadDeadline 等相关调用，当并发很大时，性能低下

解决方案：

通过 Sysfd，设置socket的读写超时

通过修改Go源码获取socket fd 修改Go 源码

社区提issue https://github.com/golang/go/issues/25729



name	old time/op	new time/op	delta	
TCP4OneShotTimeout	99.0µs ± 2%	87.9µs ± 0%	-11.20%	(p=0.008 n=5+5)
TCP4OneShotTimeout-6	18.6µs ± 1%	17.0µs ± 0%	-8.65%	(p=0.008 n=5+5)
SetReadDeadline	320ns ± 0%	204ns ± 1%	-36.14%	(p=0.016 n=4+5)
SetReadDeadline-6	562ns ± 5%	205ns ± 1%	-63.50%	(p=0.008 n=5+5)

Bytes的Buffer带来的性能问题

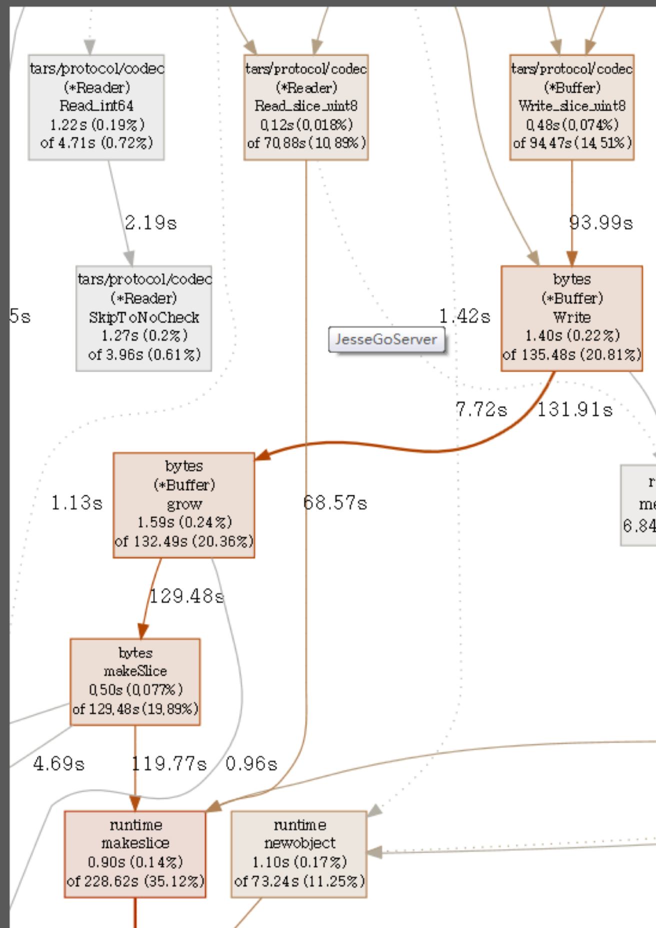
问题：

Tarsgo使用package bytes的Buffer来做编解码的流的缓冲，当Buffer不断增大时，底层的byte slice需要进行扩容，频繁的的内存分配回收对性能带来较大冲击

解决方案：

使用sync.pool来缓存临时对象，避免频繁内存分配导致的gc

后续使用sync.pool进行封装，对不同size的buffer进行分类缓存。 类似于linux 的Slab分配机制



其它优化

避免使用反射

反射有时候很好用，但往往会成为性能杀手

尽量提前确定好类型

TCP连接优化

采用长连接，设置较大的读写buffer，有选择地设置tcpNodelay

尽量避免多协程竞争chan

chan在读取和写入的时候，存在锁的开销，

当并发大时，会影响整体性能

个别场景用原子加减替代

采用协程池

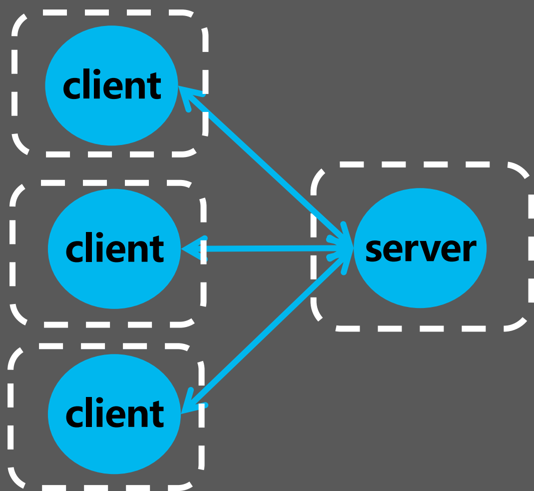
Go的协程虽然足够轻量和高效，但高并发下频繁的创建和销毁协程，还是会带来一定的性能损耗

使用指针而不是值传递

特别是大结构体，在函数设计的时候，函数参数使用指针而不是值传递，会有不错的性能提升

框架性能数据

部署情况：



测试机型：
CPU ：4核/8线程 3.30GHz
内存：16G
网卡：千兆

性能数据：

开发框架性能比较

700000
600000
500000
400000
300000
200000
100000
0

TARS(C++) TARS(JAVA) TARS(NODEJS) TARS(GO) TARS(PHP) Spring Cloud gRPC(C++) gRPC(GO)

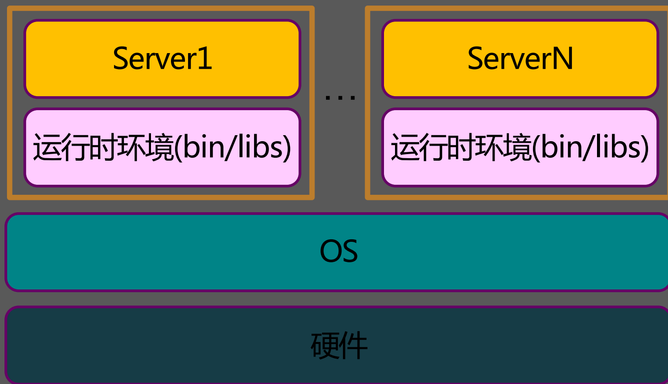
■ TPS(10字节) ■ TPS(128字节) ■ TPS(256字节)

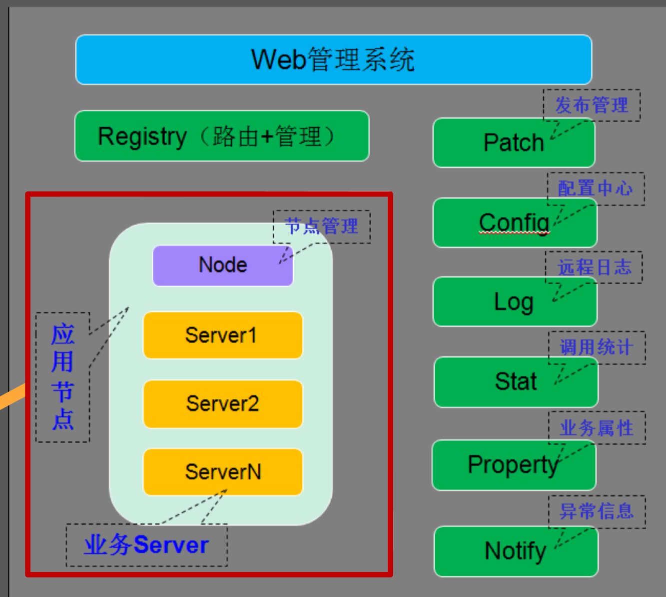
TARS集群运营曾经面临的问题

对资源的极致利用是内部运营TARS集群时最重要的原则之一

面向物理机，混合部署

- 运行时环境依赖或冲突，操作系统、基础库，环境配置等…
- 资源无法隔离，互相干扰
- 80% 的微服务CPU小于1核





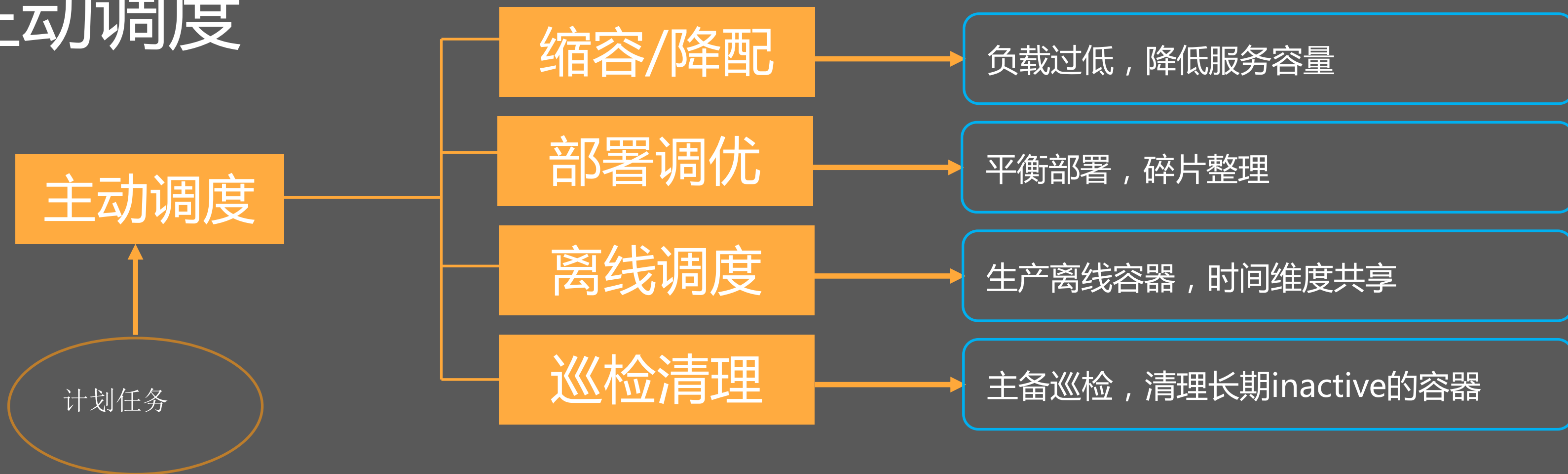
- python、jdk、glibc、openssl…
- LANG、LC_xxx
- … …

TARS on Docker

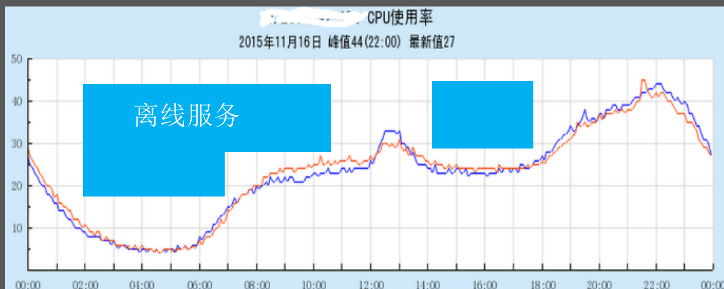
TARS天生自带路由服务，解决服务注册和发现问题，应用程序不直接面对IP、端口

借助调度中心实现业务完全托管
使用Host网络模式
使用cpu.cfs_quota_us 限制 ，最小1个ECU（0.1核）

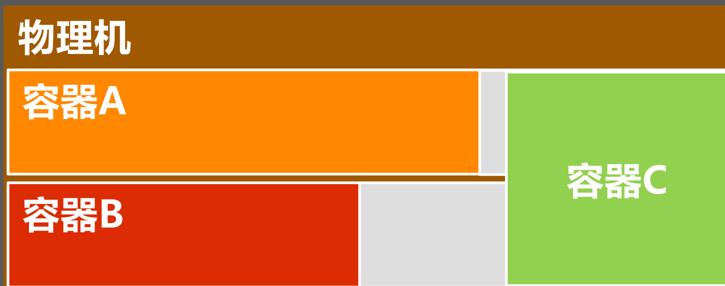
主动调度



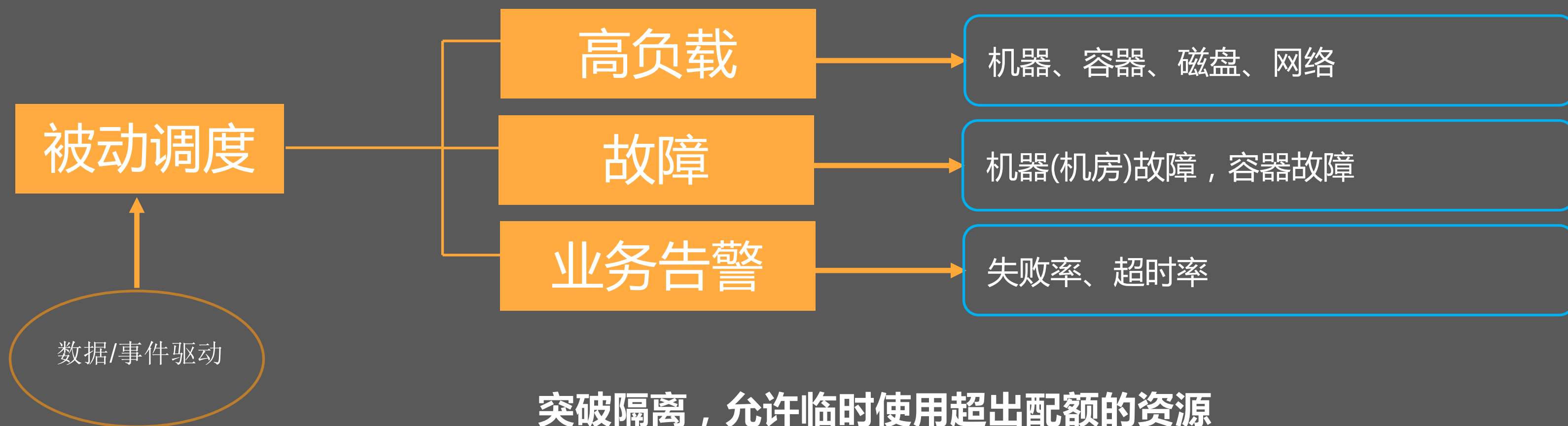
时间维度上的共享



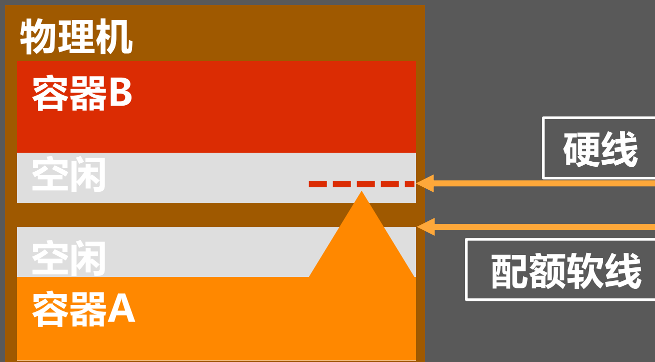
超卖



被动调度



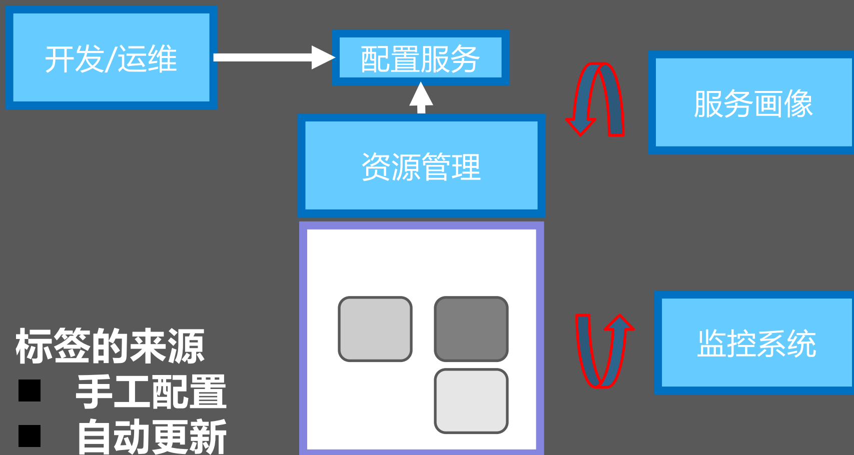
突破隔离，允许临时使用超出配额的资源



标签系统

各种部署要求层出不穷
- **要(不要）什么机器？**
- **要(不要）和什么服务一起部署？**
- **只能部署(x个)什么样的服务？**
- **不能部署什么样的服务？**
- **...**

prefer/forbid ， belong/own
在服务和机器维度标记各自的属性和偏好，通过标签匹配实现特定部署要求



标签的来源
- **手工配置**
- **自动更新**

大规模实战中的镜像传输

Layer1
Layer2
Layer3

合并

压缩上传

仓库
（深圳/上海/天津）

Ceph

P2P

宿主机
P2P Client

宿主机
P2P Client

宿主机
P2P Client

镜像压缩，P2P传输
使用 Golang编写docker的http proxy，用p2p下载

镜像里不放配置数据
镜像里不放业务的应用数据，尤其是体积大、变更频繁的类型，比如地图路况数据

Tars On Kubernetes

Web平台

Open API

AdminRegistry

1.交给k8s操作容器
2.通过watch实现k8s
到tars的数据同步

k8s集群

Master节点

API Server

Worker节点

Pod
容器
业务进程
TarsNode进程

Pod
容器
业务进程
TarsNode进程

Kubelet

docker

Worker节点

Pod
容器
业务进程
TarsNode进程

Pod
容器
业务进程
TarsNode进程

Kubelet

docker

TARS应用案例

Tars在腾讯内部使用十年，并于2017年4月10日开源，开源后与业界众多企业交流，同时也得到了广泛应用。
开源地址： https://github.com/TarsCloud
2018年6月加入Linux 基金会
Go语言开源： https://github.com/TarsCloud/TarsGo

扫码加入TarsGo 交流群

扫码添加TARS小助手为好友，拉你入群

（PS：申请时请备注“**GC Con**”）