猫眼电影

Chronus分享

- 分布式任务调度系统背景知识
- Chronus的设计与实现
- 未来规划

- 分布式任务调度系统背景知识

- 什么是分布式任务调度系统?

- 为什么需要分布式任务调度系统?

- 目前开源系统对比

什么是任务?

实现 **业务功能** 的 代码集合

更新缓存

数据对比

拉取数据

Java/PHP/Python/Go

脚本

必须 **触发** 才可执行

时间触发

事件触发

什么是调度?

每天凌晨2点0分0秒

刷新商城首页缓存

什么是分布式任务调度系统?

把分散的，可靠性差的计划任务纳入统一的平台，并实现集群管理调度和分布式部署的一种定时任务的管理方式。

为什么需要分布式任务调度系统?

- 支付系统每天凌晨1点跑批，进行一天清算，每月1号进行上个月清算
- 电商整点抢购，商品价格8点整开始优惠
- 12306购票系统，超过30分钟没有成功支付订单的，进行回收处理
- 保险人管系统每月工资结算，有150万代理人，如何快速的进行工资结算(数据运算型)

为什么需要任务调度系统?

解决具有以下特征的问题：

- 时间驱动/事件驱动：内部系统一般可以通过事件来驱动，但涉及到外部系统，则只能使用时间驱动。如：抓取外部系统价格。每小时抓取，由于是外部系统，不能像内部系统一样发送事件触发事件。

- 批量处理/逐条处理：批量处理堆积的数据更加高效，在不需要实时性的情况下比消息中间件更有优势。而且有的业务逻辑只能批量处理，如：电商公司与快递公司结算，一个月结算一次，并且根据送货的数量有提成。比如，当月送货超过 1000 则额外给快递公司多 1% 的快递费。

- 非实时性/实时性：虽然消息中间件可以做到实时处理数据，但有的情况并不需要。如：VIP 用户降级，如果超过 1 年无购买行为，则自动降级。这类需求没有强烈的时间要求，不需要按照时间精确的降级 VIP 用户。

目前开源系统对比

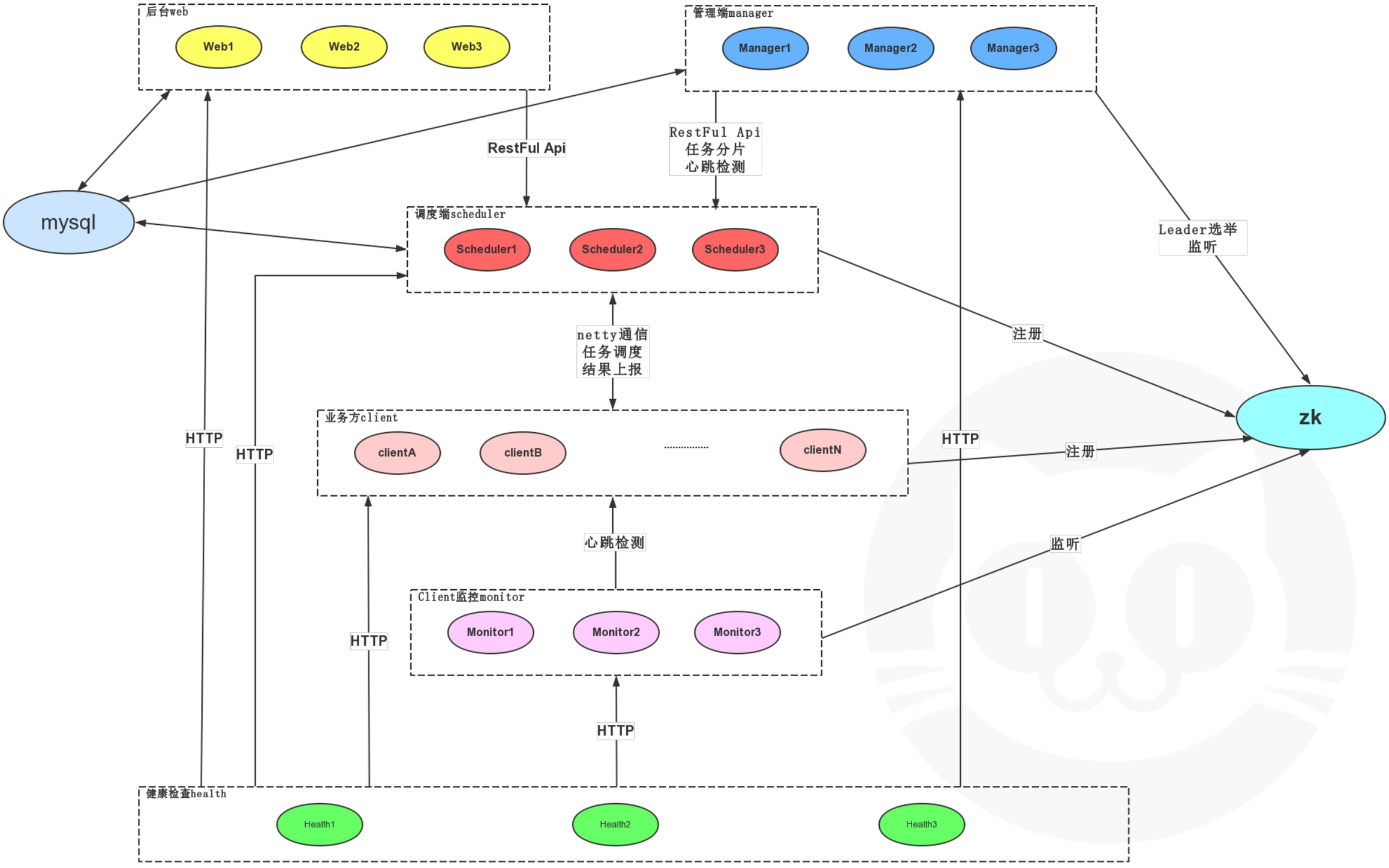
feature	cronsun	Elastic-job	Saturn	XXL-Job	Chronus
项目背景	替代crontab	当当网开源，文档齐全	唯品会自主研发，基于当当Elastic-job版本1	大众点评，文档齐全	美大
语言	Go	Java	Java	Java	Java
依赖	etcd、mongodb	jdk7+、zk、maven、mesos、mysql	jdk7+、maven、nodes.js、npm、docker	jdk7+、maven、mysql	jdk7+、maven、nodes.js、npm、mysql
HA		支持	支持	支持	支持
弹性扩容	null	支持 下次作业前重新进行分片，不影响当前作业执行	支持 支持容器化技术自动扩容缩容，保证高峰期处理的弹性架构	支持 一旦有执行器上线下线，下次调度会重新分配任务	支持 下次作业前重新进行分片，不影响当前作业执行
分片广播	null	支持	支持	支持	支持
动态分片	null	支持	支持	支持	支持
易用、易部署	安装简单、使用简便	易用易部署	可以很好的部署在docker容器上	支持通过web页面对任务进行CURD操作、操作简单、一分钟上手	易用易部署
Web UI（任务/日志/报表）	支持	支持	支持	支持	支持
任务监控		支持	支持	支持	支持
日志可追踪	有日志查询界面	有日志查询界面，job的运行日志会记录到数据库内	支持	支持在线查看调度结果，并且支持以Rolling的方式实时查看输出的完整执行日志	支持在线查看调度结果，并且支持以Rolling的方式实时查看输出的完整执行日志
失败告警	支持	支持	支持	支持	支持
多语言任务（Python、Shell）	不支持	支持	支持	支持	不支持
高级功能	任务失败自动重试	失效转移、错过执行作业重触发、自诊断并修复分布式不稳定造成的问题、支持并行调度、丰富的作业类型、Spring整合以及命名空间提供	失效转移、超时控制、容器化支持、JMX健康检查、多zk集群、资源动态平衡	路由策略、故障转移、阻塞处理、超时控制、失败重试、任务依赖、国际化、容器化	

Chronus的架构与实现

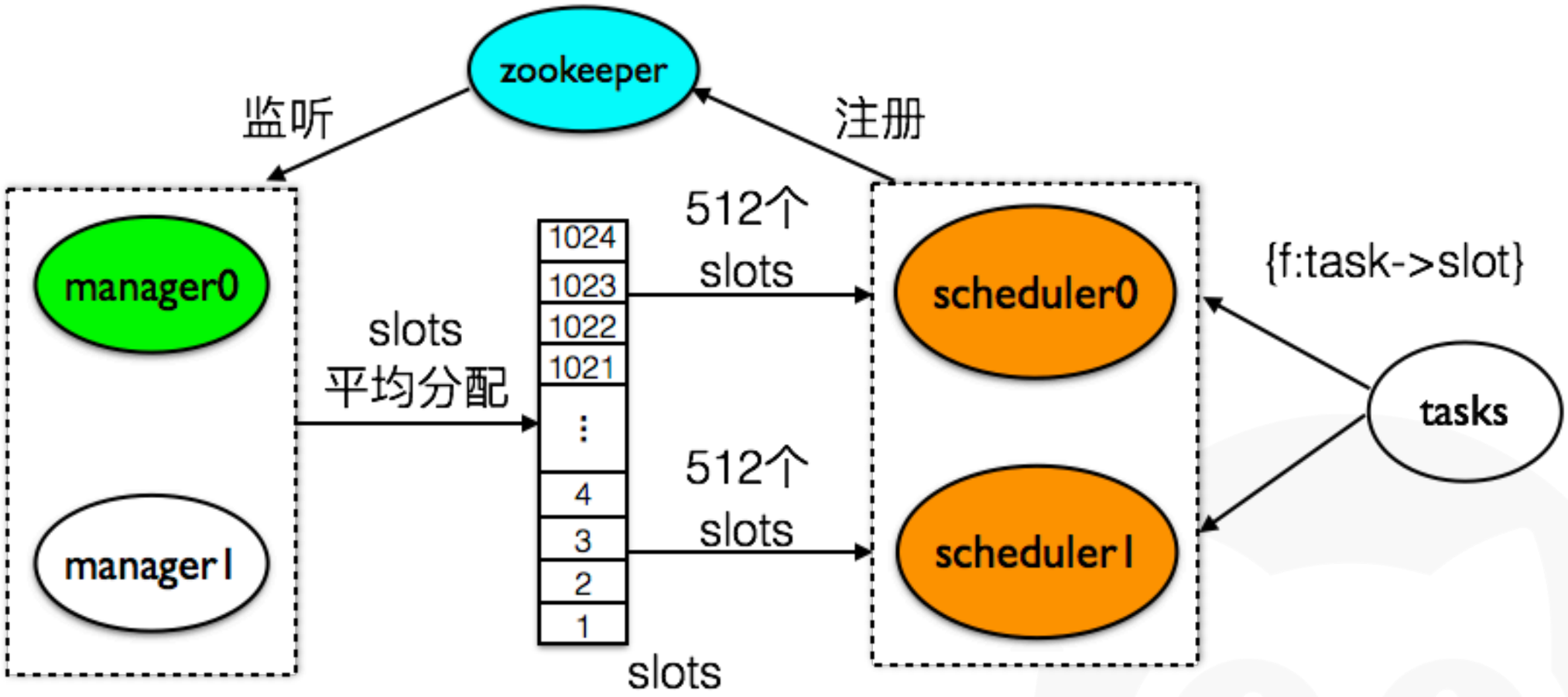
- 整体架构图

- 核心功能设计介绍

整体架构图


后台web
Web1
Web2
Web3
管理端manager
Manager1
Manager2
Manager3
RestFul Api
RestFul Api
任务分片
心跳检测
mysql
调度端scheduler
Scheduler1
Scheduler2
Scheduler3
Leader选举
监听
netty通信
任务调度
结果上报
注册
HTTP
HTTP
业务方client
clientA
clientB
.....
clientN
HTTP
注册
zk
心跳检测
监听
Client监控monitor
Monitor1
Monitor2
Monitor3
HTTP
HTTP
健康检查health
Health1
Health2
Health3

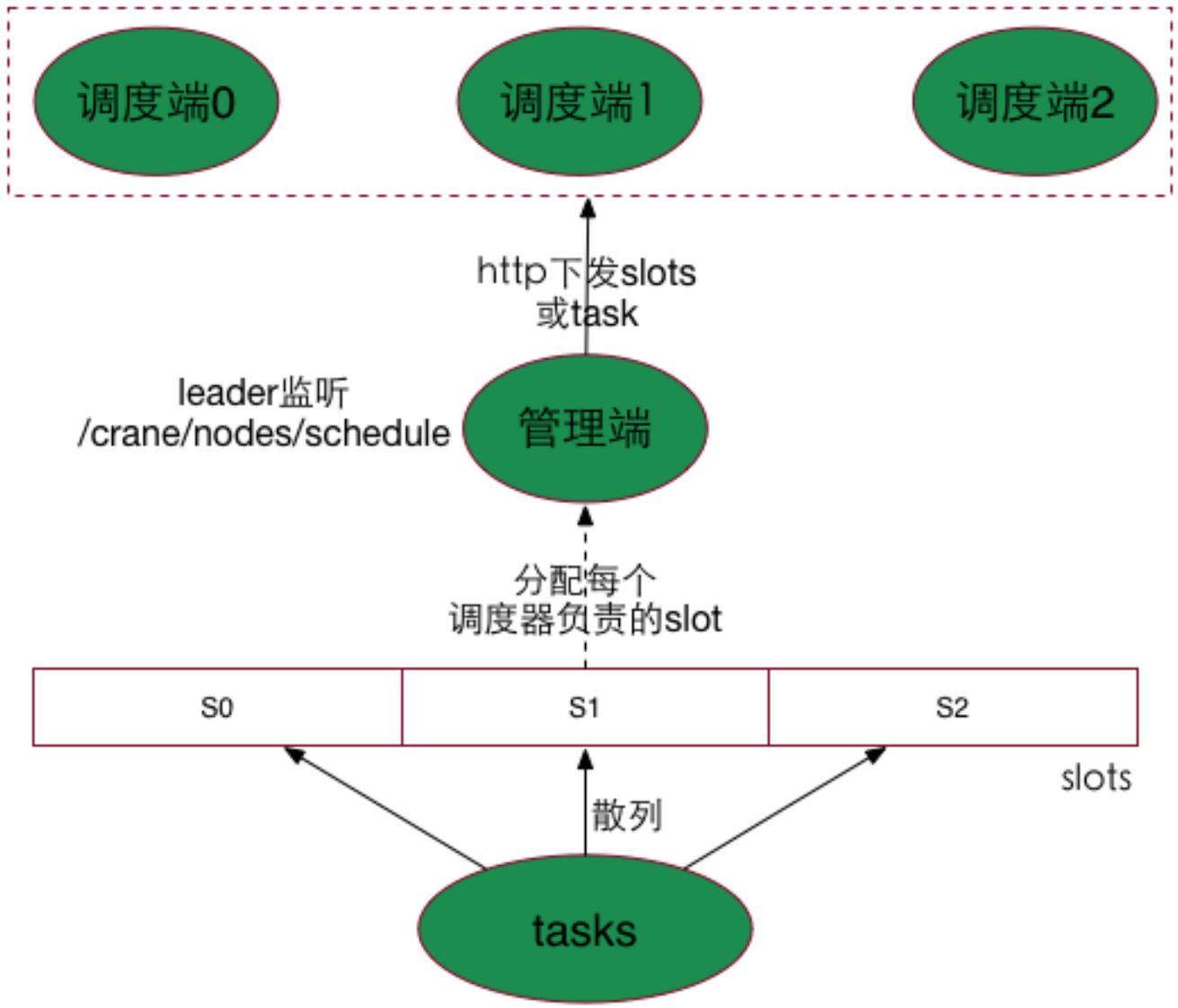
核心功能设计介绍-调度模型



管理端集群通过zk选举出一个leader负责slot的计算和与调度端的交互。管理端负责将预先分配好的slot写入zk（第一次通过手动触发写入zk），并监听路径/chronus/nodes/scheduler，获取调度端的上下线状态。

1. 管理端首先会给每个调度端分配负责的slots，并持久化到zk路径/chronus/nodes/slots。
2. 每个slot对应的调度器持久化到zk路径/chronus/nodes/slots/n（n=1至1024）下
3. 调度端启动后会从zk读取自身负责的slots，开始调度。

核心功能设计介绍-调度模型

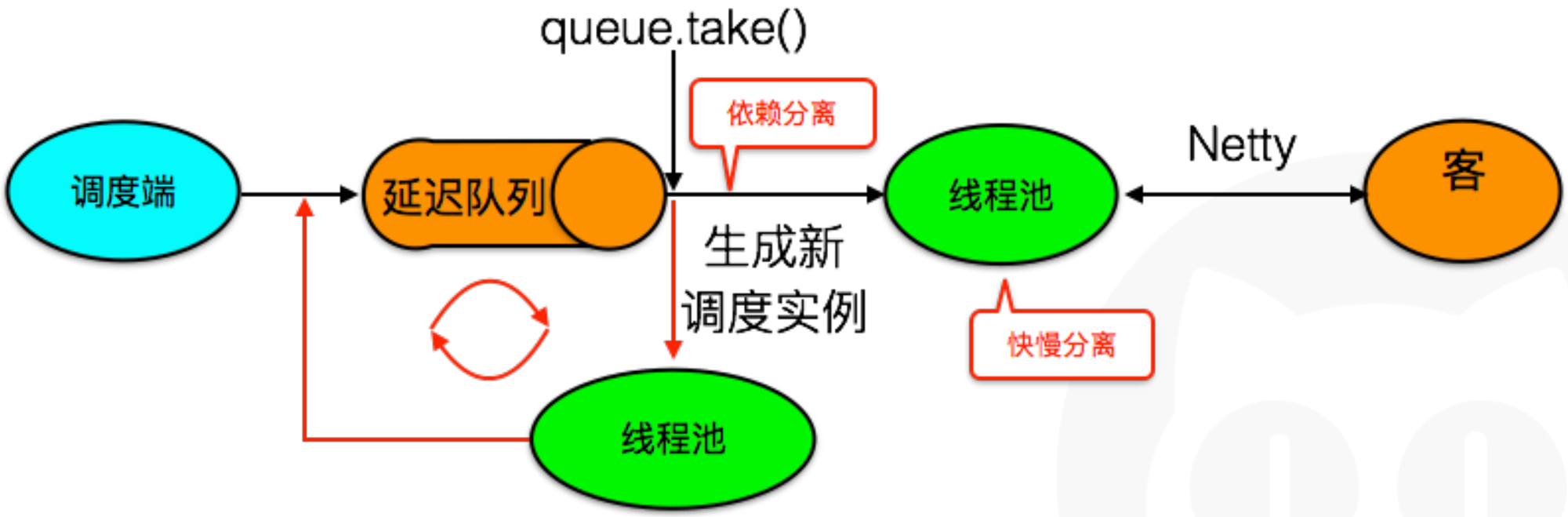


问题：为什么要引入slots层?

1. 任务是在时刻动态变化，如果通过直接将任务分配给调度器，就是对一个变量进行操作，增加任务分配的复杂度。
2. 通过引入slots中间层，将任务散列到slot，而slots大小是固定的(Chronus初始化大小为1024)，这样通过将slot分配给调度器，间接的实现了对任务的分配。
3. slots是通过管理端分配的，管理端通过监听ZK路径/chronus/nodes/schedule获得调度器增加和减少事件，从而将slots平均分配给每个调度器。

核心功能设计介绍-调度模型

调度引擎



核心功能设计介绍-调度模型

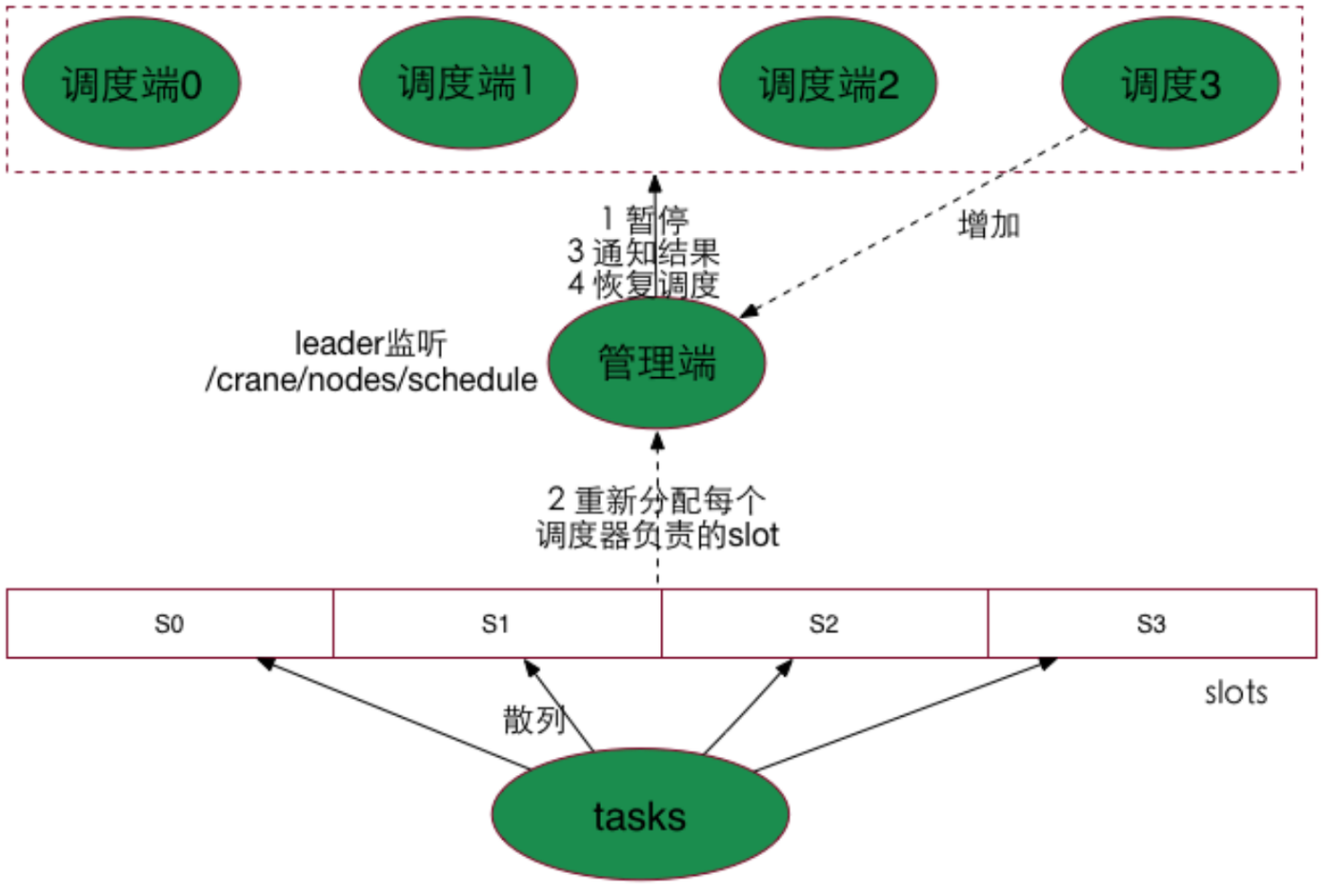
管理端扩容缩容?

管理端扩容缩容时会导致leader重新选举，不影响调度端任务调度。

万一管理端全部挂了怎么办?

核心功能设计介绍-调度模型

调度端扩容缩容?



1. 调度端扩容时，zk会通知管理端leader。此时会触发slots重新分配。
2. 管理端首先计算出每个调度器负责的slots，通过http先暂停调度，更新每个调度器负责的slots。
3. 当所有调度器都更新完毕并回复管理端后，管理端再通过http恢复调度器的调度工作。

核心功能设计介绍-调度模型

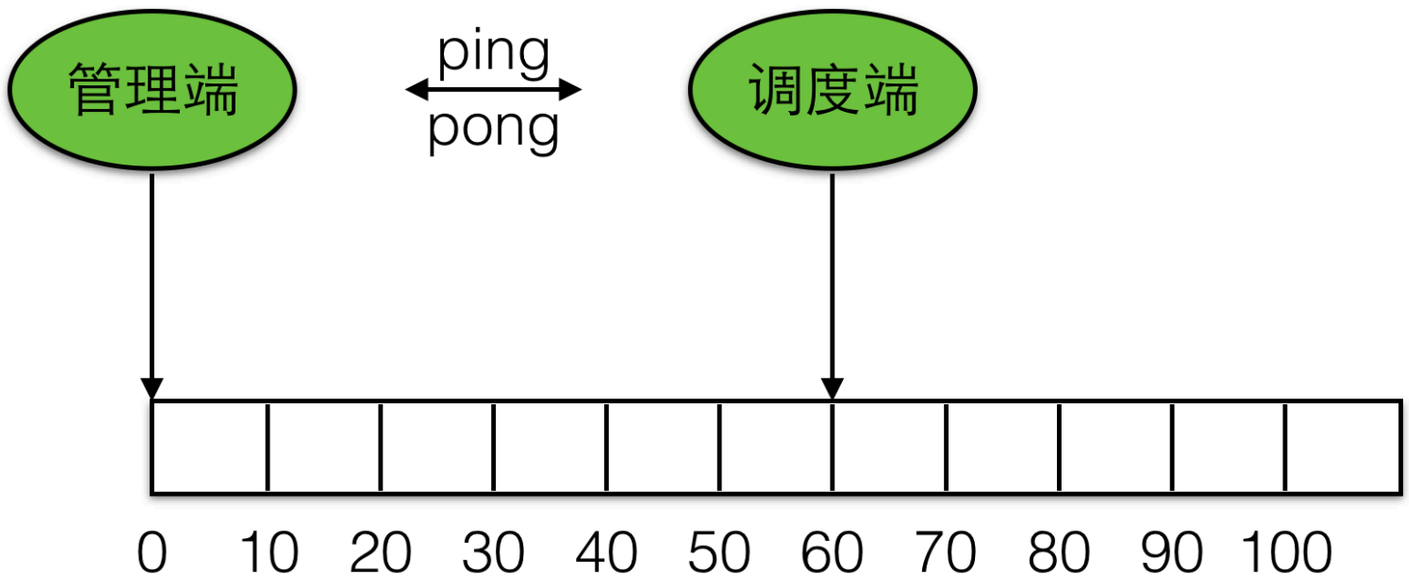
问题1：目前scheduler注册和manager监听的zk节点均为持久节点，万一scheduler出问题，没有删除/chronus/nodes/schedule/ip节点怎么办?

manager心跳检测

问题2：万一只是单纯的网络抖动，实际上scheduler是正常的?

核心功能设计介绍-调度模型

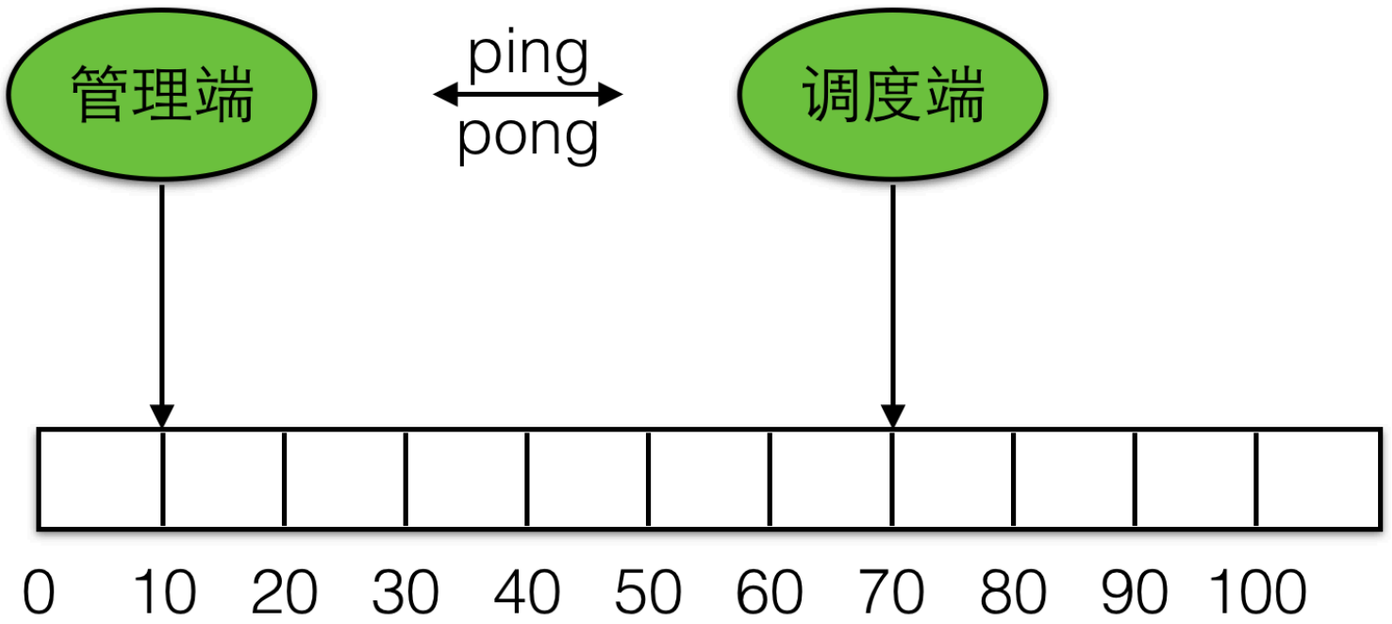
租约模型



横坐标表示时间，0表示当前时间，每个桶的间隔是10秒钟，
也就是每隔10秒钟ping一下调度端，调度端回复pong。

核心功能设计介绍-调度模型

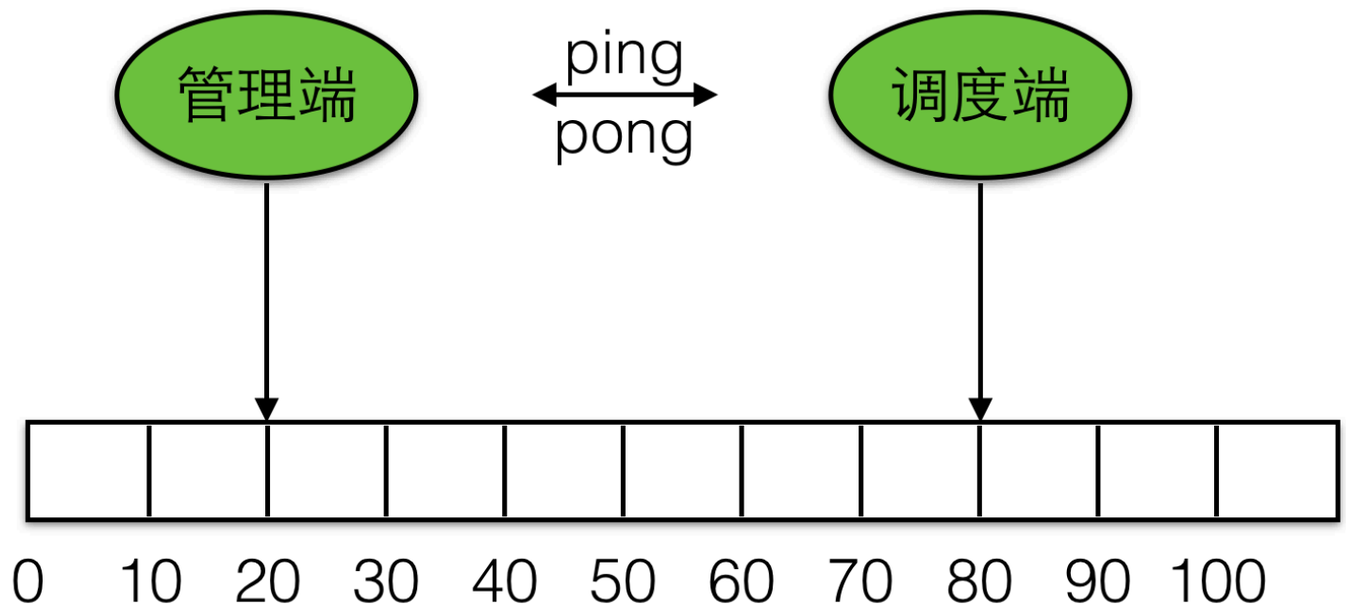
租约模型



调度端ping一次后会沿桶移动10秒钟，如果调度端回复pong，则也沿桶移动10秒钟。

核心功能设计介绍-调度模型

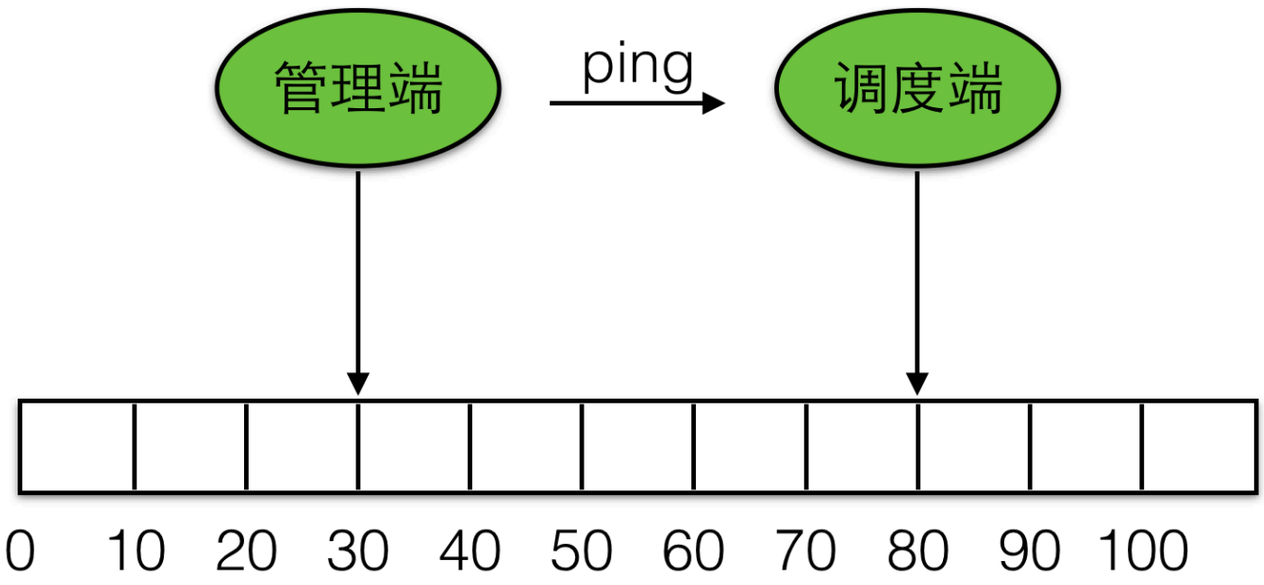
租约模型



如果管理端和调度端一切正常，则管理端与调度端的时间间隔则一直为调度端的租期，此处为60秒钟。

核心功能设计介绍-调度模型

租约模型



如果调度端没有回复pong，则管理端会逐渐逼近调度端，知道追赶到，然后检测到调度端宕机，将任务进行fail-over。

核心功能设计介绍-调度模型

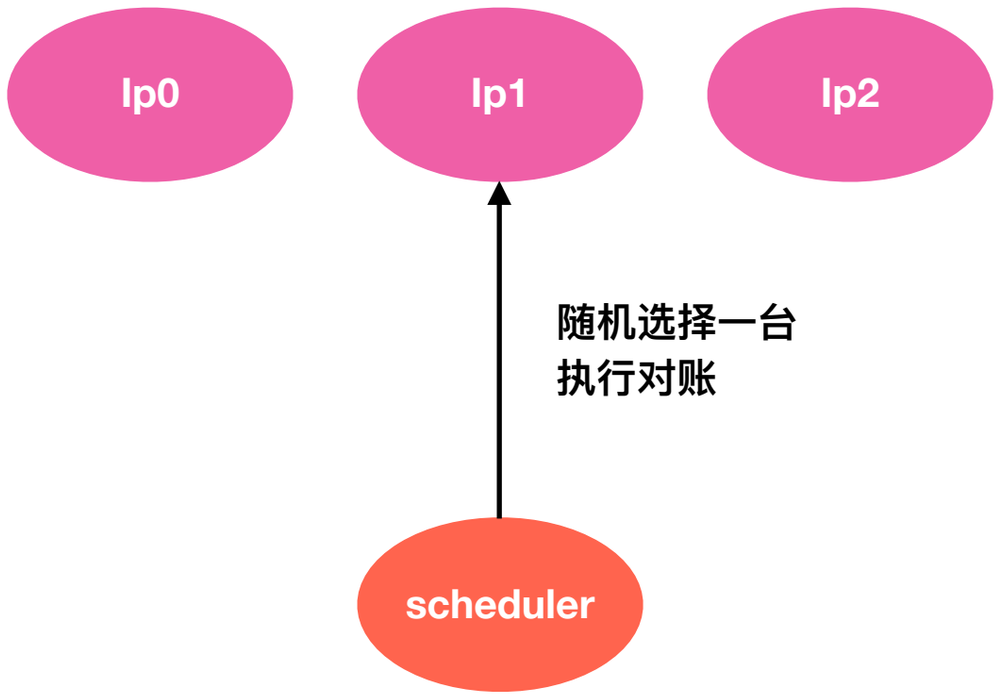
租约模型

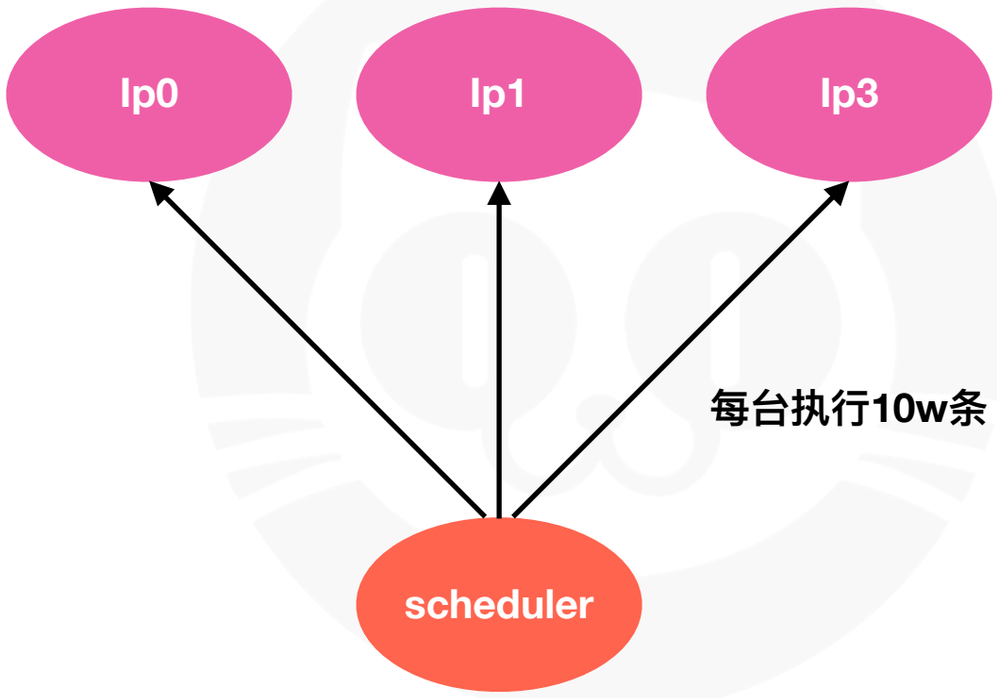
1. 调度端启动后在Zookeeper路径/chronus/nodes/scheduler上注册持久节点，管理端监听到节点添加事件后进行slots重新分配。
2. 调度端宕机后，管理端心跳检测在租期60秒钟没有收到任务回复后，重新分配slots，管理端会删除Zookeeper路径/chronus/nodes/scheduler中该调度器的持久节点。
3. 调度端重启时，在租期内重启，心跳检测正常，不会触发slots重新分配。【尽量减少slots，保证服务可用性】
4. 调度端重启时，如果超过60秒，会触发slots重新分配，重启成功后会再次出发slots进行分配。
5. 租期在配置中心可配置，可以随时更改，如果遇到管理端全部宕机的情况，可以将租期设置调大。

目前心跳检测存在于：
manager -> scheduler
monitor -> client

核心功能设计介绍-任务分片

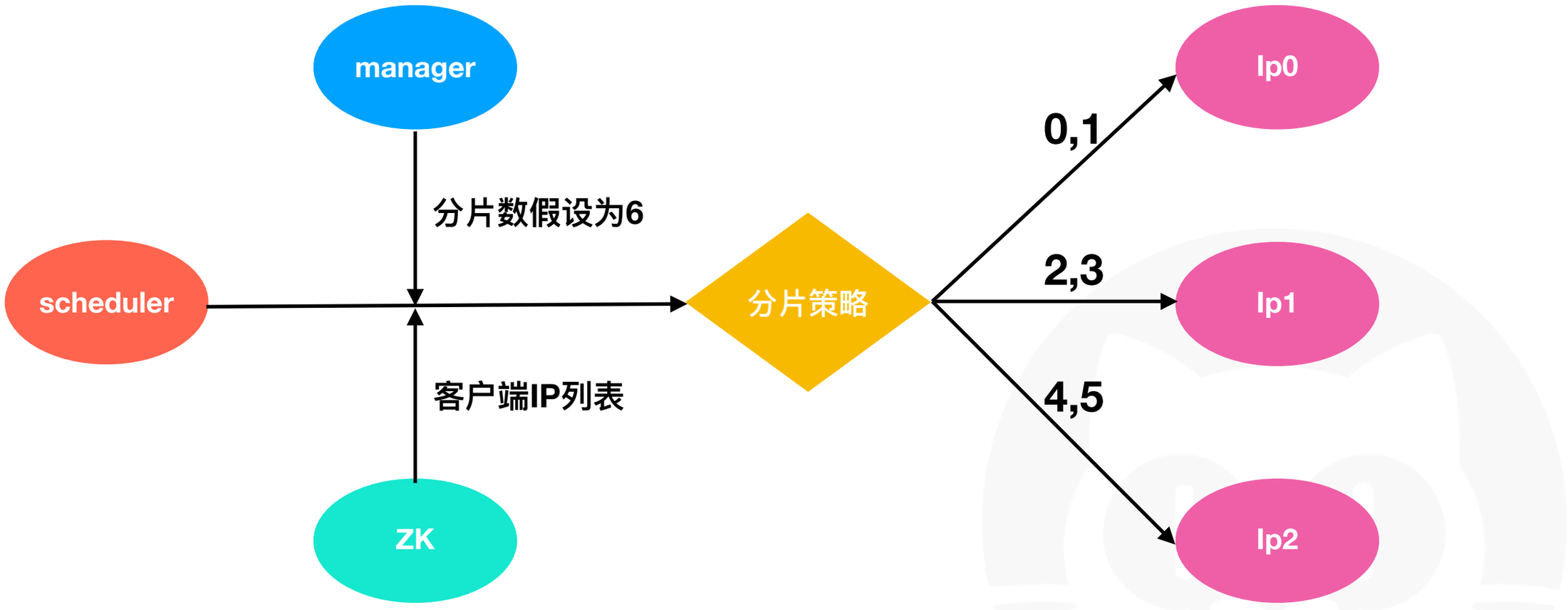
问题：有个定时任务需要处理30w条对账数据。小明的应用部署在集群中3台机器上，如果每次只让其中一台处理全部30w条对账数据，并不能充分利集群中的其他资源。





核心功能设计介绍-任务分片

解决方案



核心功能设计介绍-任务分片

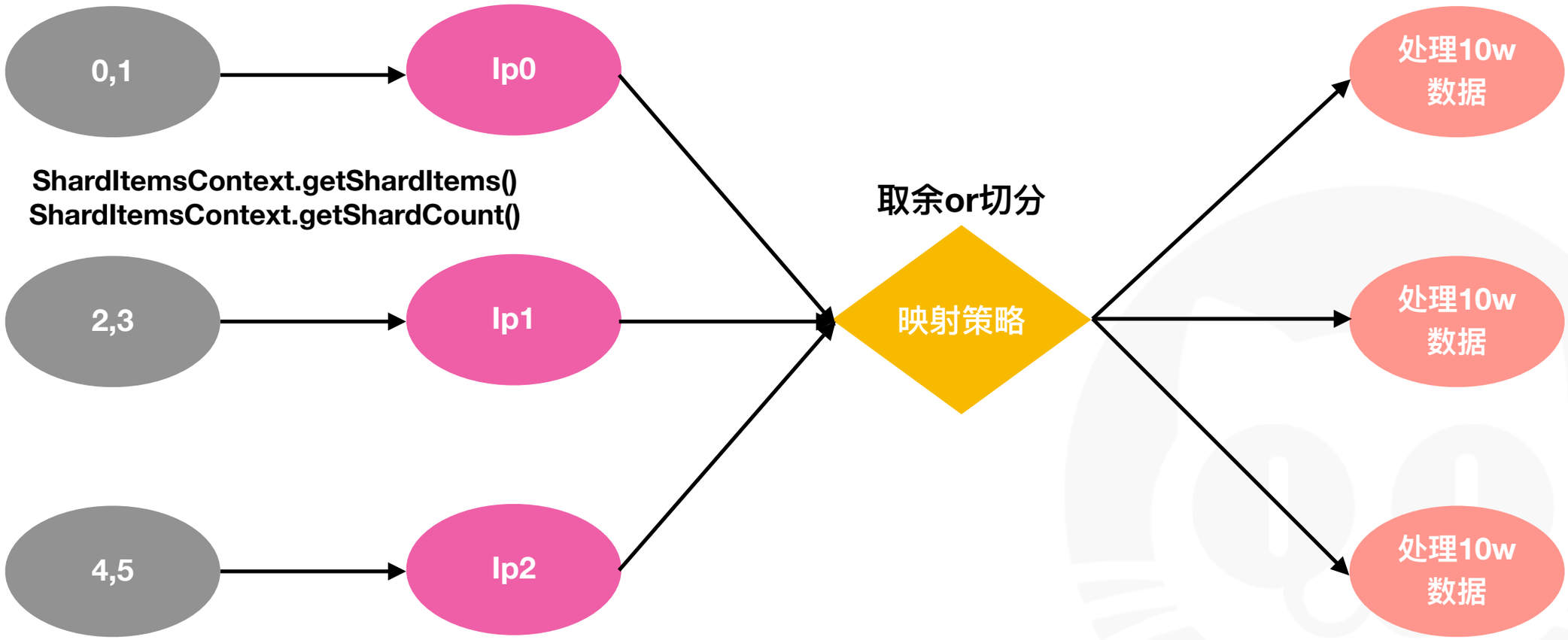
解决方案

目前分片逻辑仅支持平均分片
首先对客户端IP进行排序，如果机器个数不能整除分片总数， 则多余分片将依次追加排序较低的IP。

分片数/客户端	Ip0	Ip1	Ip2
5	0,3	1,4	2
6	0,1	2,3	4,5
7	0,1,6	2,3	4,5

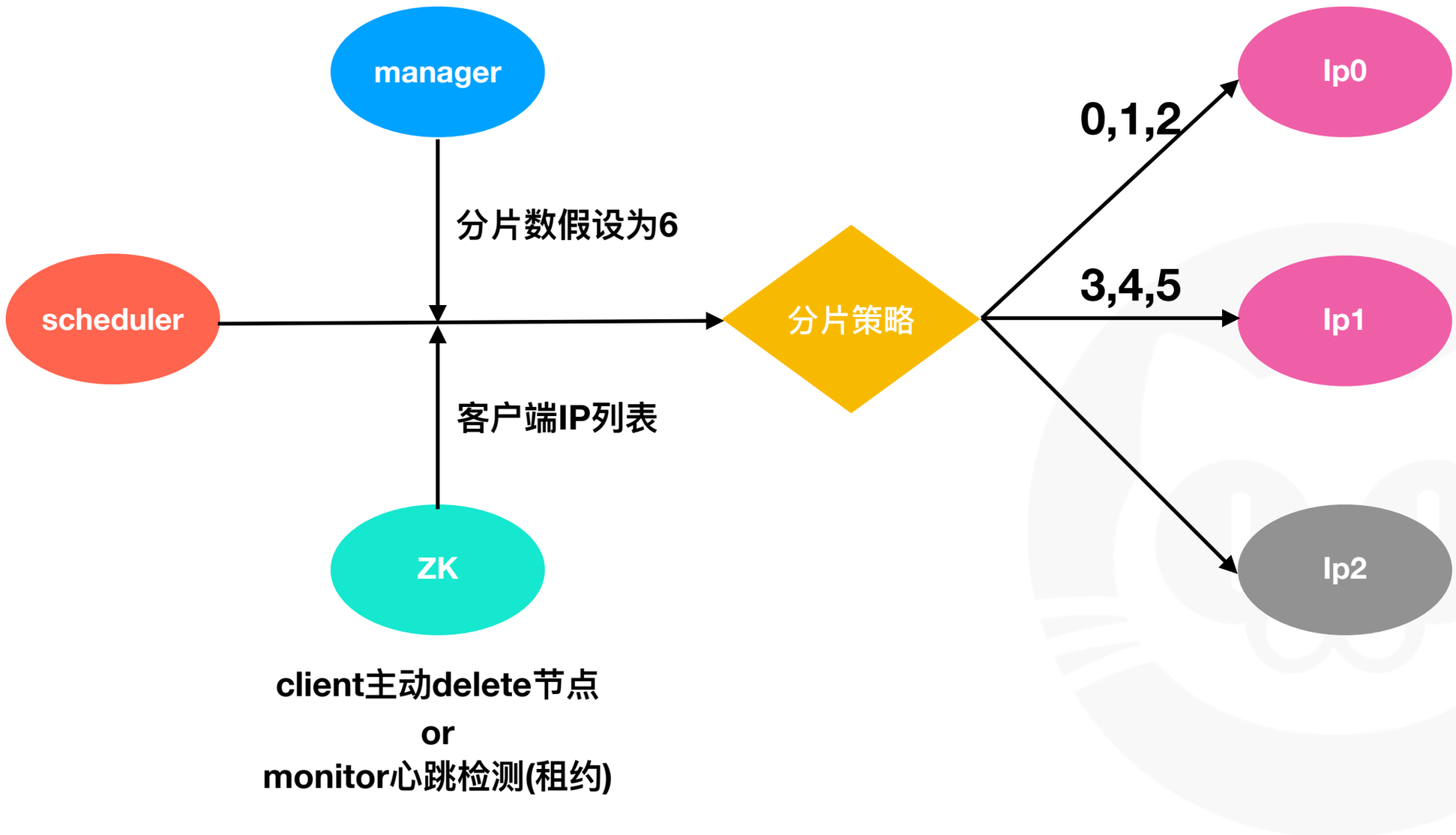
核心功能设计介绍-任务分片

业务方解决方案



核心功能设计介绍-任务分片

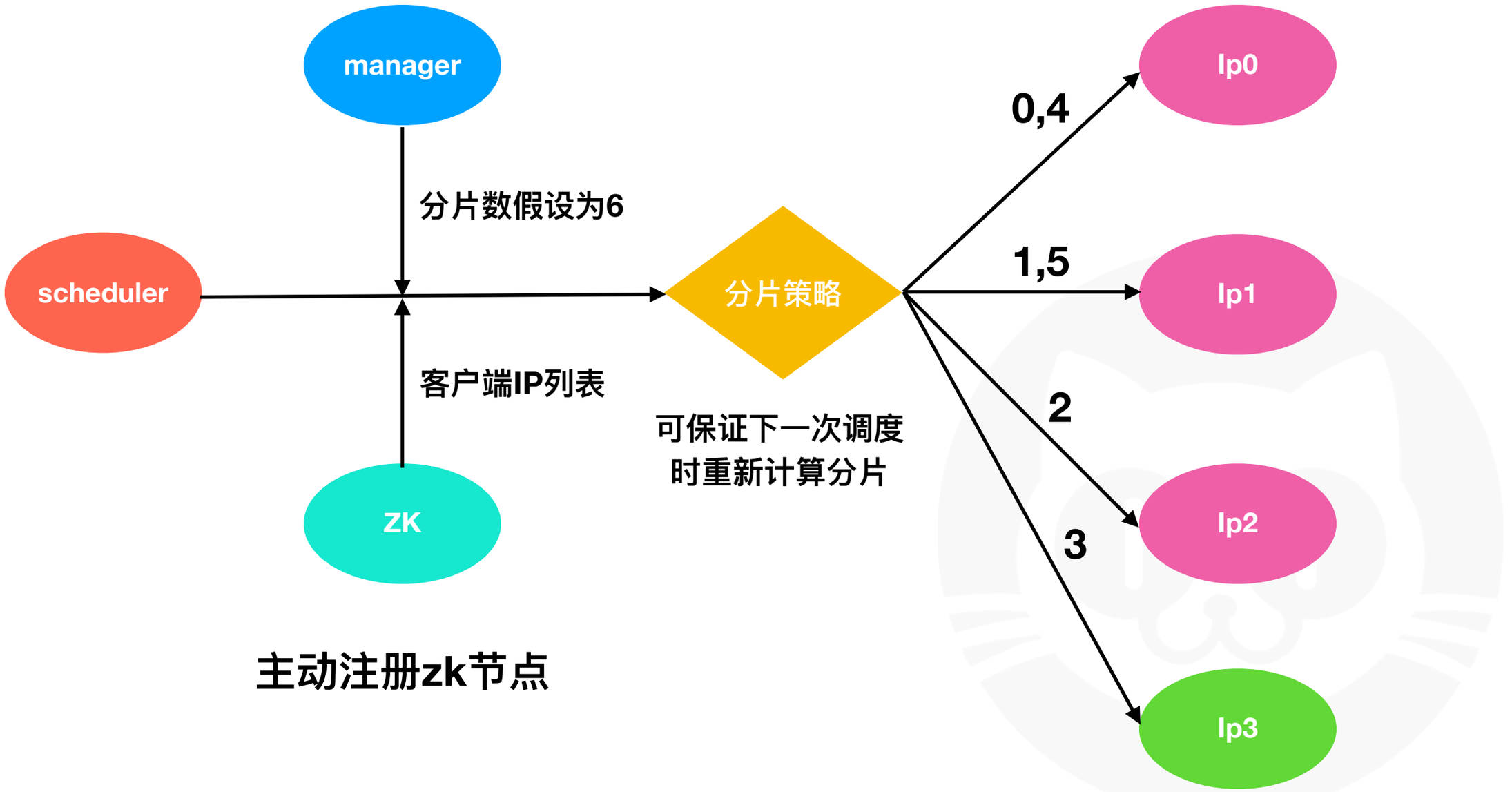
客户端宕机



client主动delete节点
or
monitor心跳检测(租约)

核心功能设计介绍-任务分片

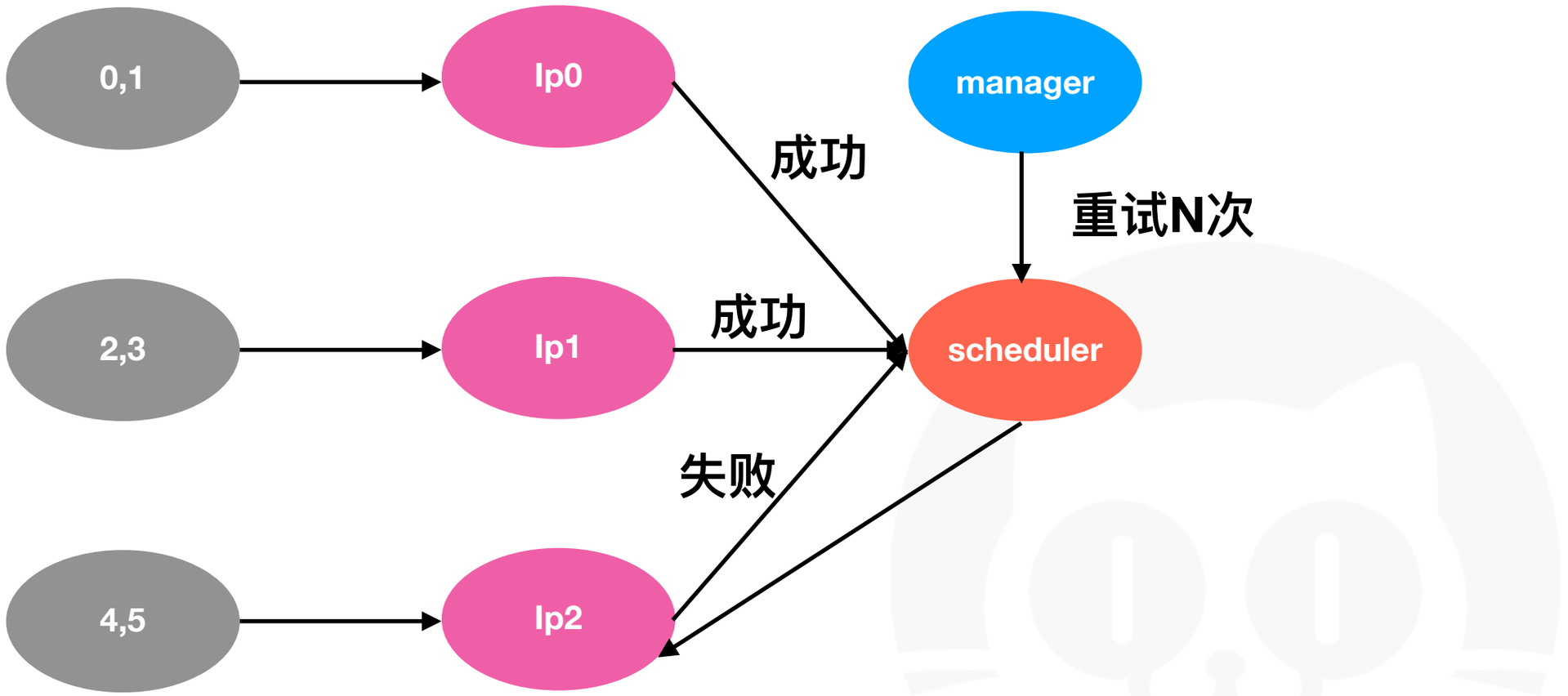
客户端扩容



主动注册zk节点

核心功能设计介绍-任务分片

失败重试



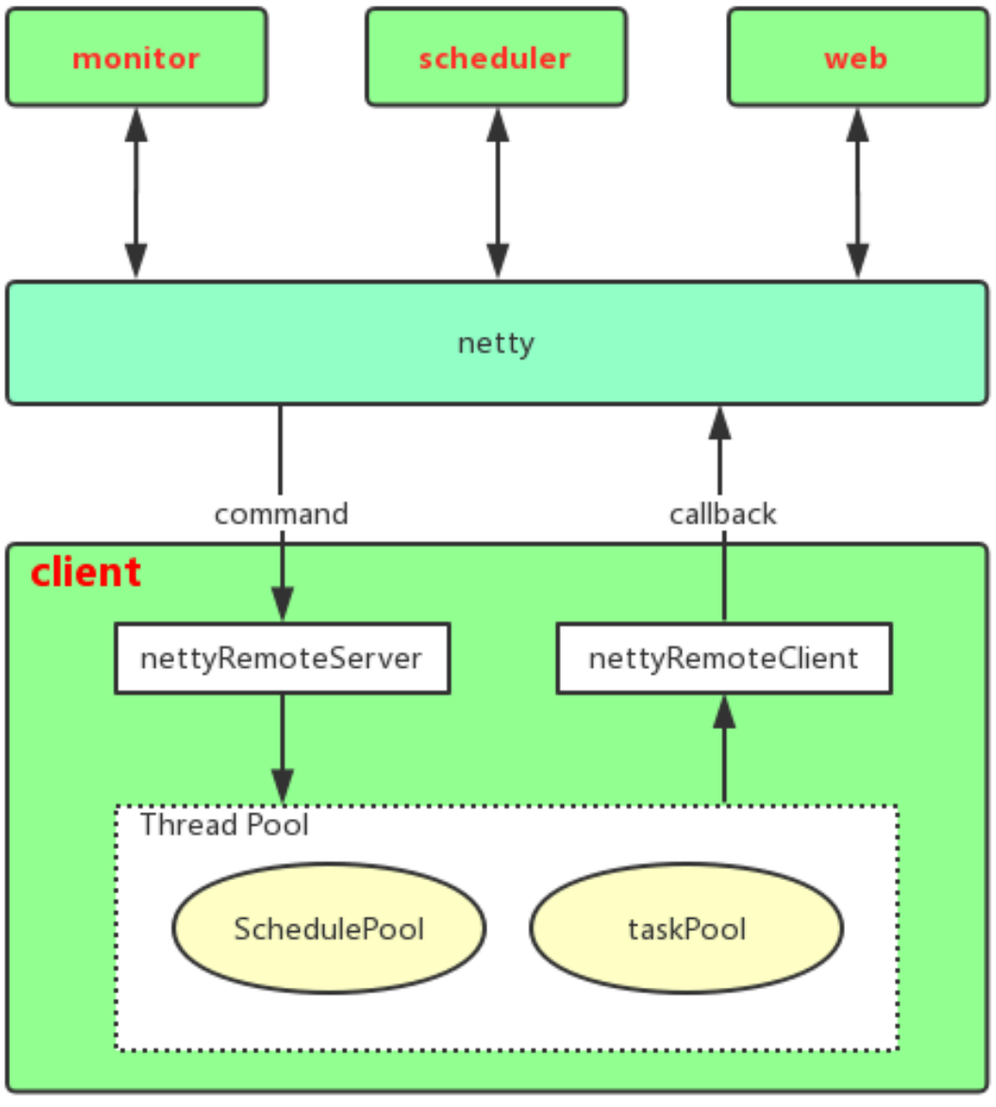
核心功能设计介绍-client

问题：client核心需求是什么？

接受调度端的任务执行请求并”可靠的”发送回执给调度端

核心功能设计介绍-client

交互流程



核心功能设计介绍-client

如何可靠的发送回执给调度端?

解决途径:

Zookeeper persistent： node /chronus/heartbeats/client

Netty client/server

最大限度上报（重试&重启hook）

未来规划

系统整体业务健康度检测和评估手段改进

1. 三个层面的监控并不够：硬件指标、系统和进程、组件和链路
2. 加强系统监控综合评估能力

自动测试体系完善

1. 单元测试不足以发现大流量负载，复杂并发场景下的隐蔽bug
2. 需要构建随机生成测试用例和模拟组件失效模式的测试体系

功能扩展

1. 增加系统HA能力
2. 。。。

Thanks!