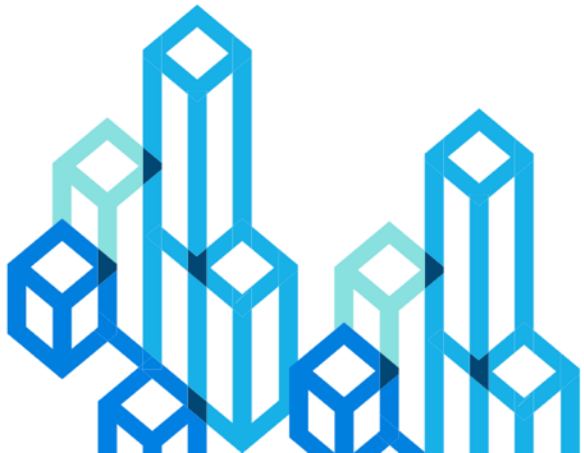
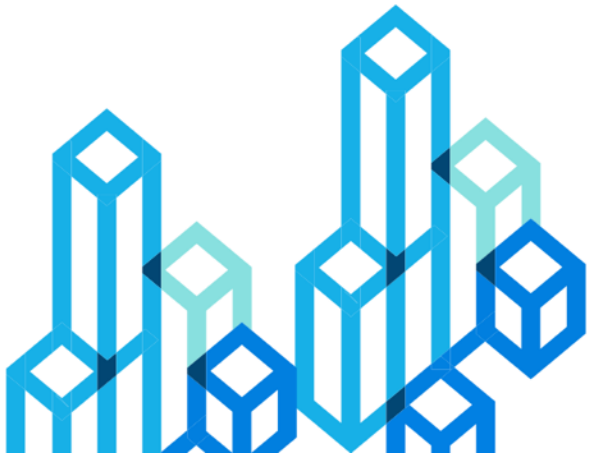
1W节点K8S集群的
云原生混部运营与运维实践

星龙





目录

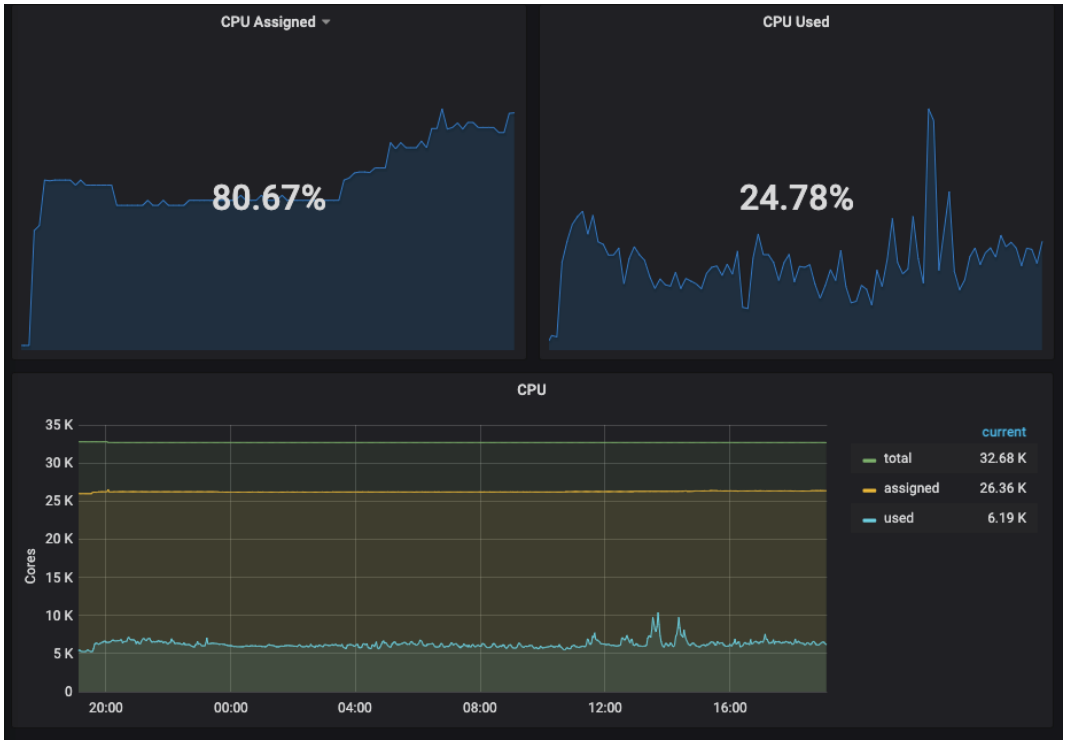
01 运营：系统性提高资源利用率

02 运维：大规模 kubernetes 集群建设

01

运营：系统性提高资源利用率

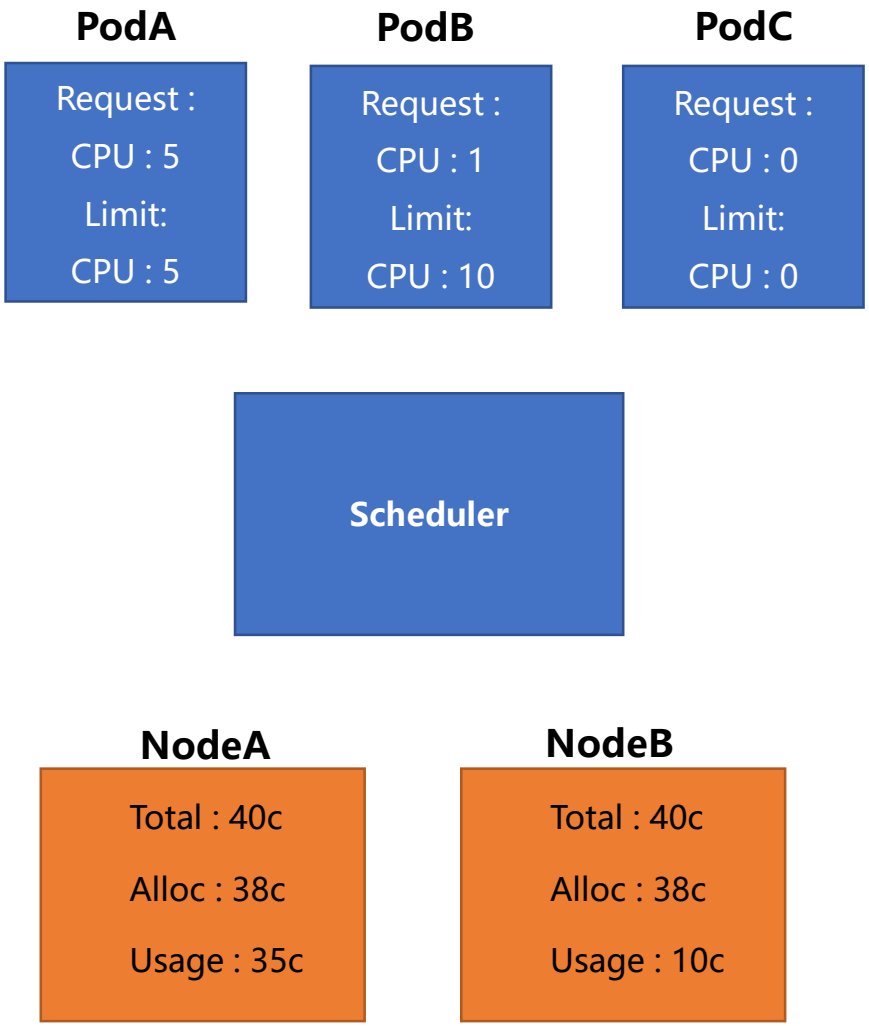
1.1. 构建原生 Kubernetes 集群 – 初探



- 800+ 节点
- 分配率约80%
- 使用率不足25%

这一部分资源还能再用吗??

1.2. 为什么原生 Kubernetes 无法解决资源利用率的问题



PodA :

- 无法被调度
- NodeB 上仍有空闲资源

PodB :

- 可以被调度到 NodeA 或 NodeB
- 当调度到 NodeA 时可能被驱逐

PodC :

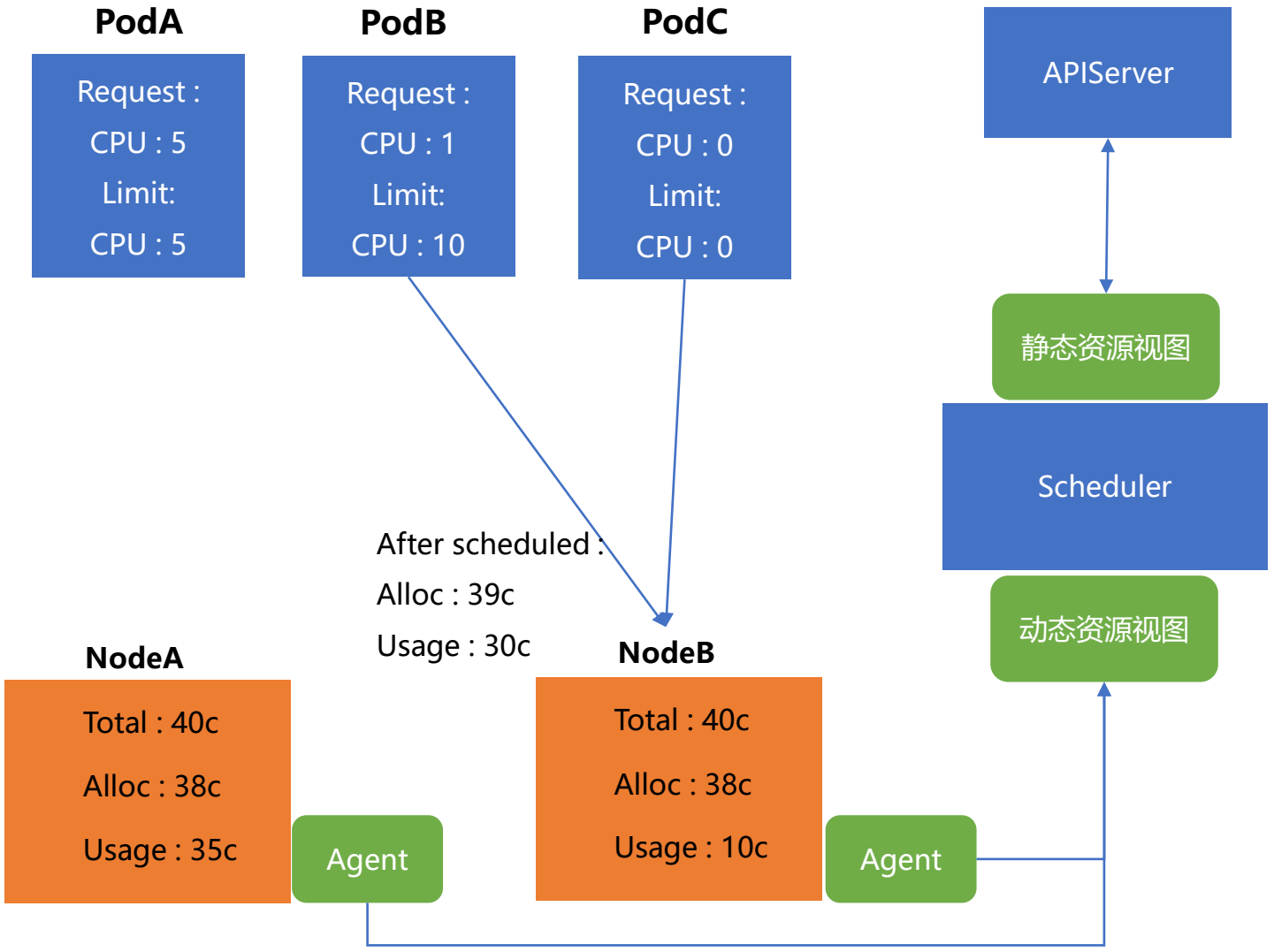
- 可以被调度
- 当调度到 NodeA 时可能被驱逐

根本原因：

资源使用是动态的，而配额是静态限制

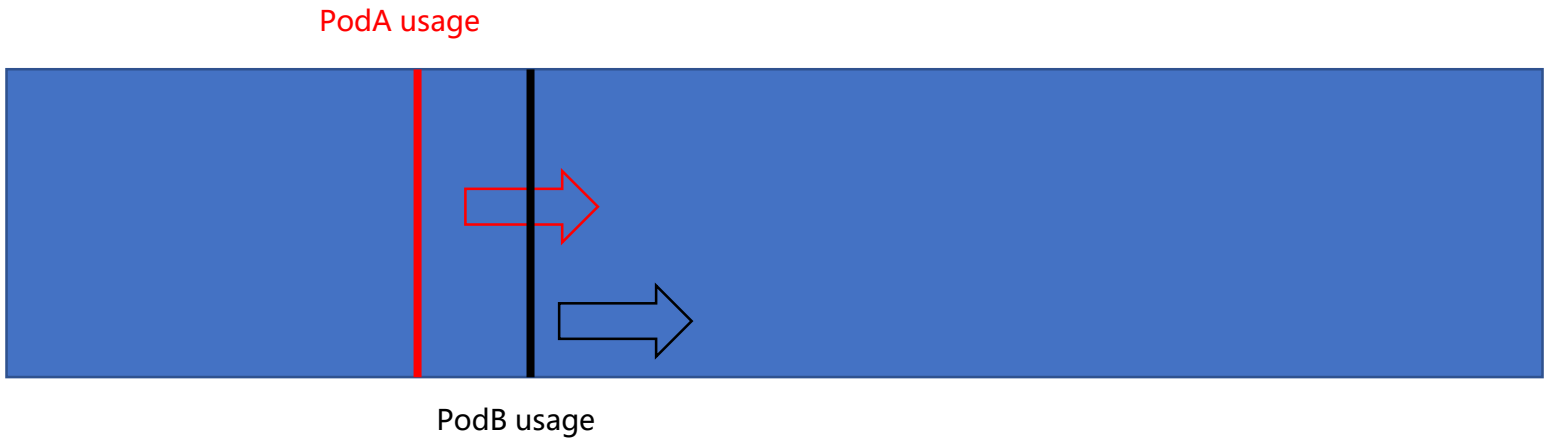
原生调度器并不感知资源使用情况

1.3. 引入动态资源视图

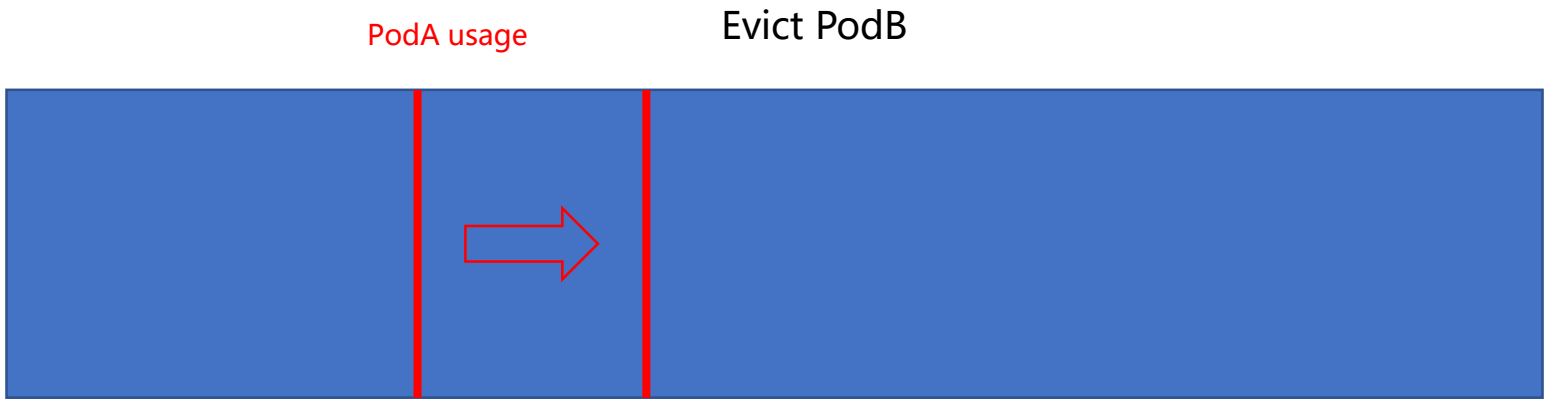

PodA
Request :
CPU : 5
Limit:
CPU : 5
PodB
Request :
CPU : 1
Limit:
CPU : 10
PodC
Request :
CPU : 0
Limit:
CPU : 0
APIServer
静态资源视图
Scheduler
动态资源视图
After scheduled :
Alloc : 39c
Usage : 30c
NodeA
Total : 40c
Alloc : 38c
Usage : 35c
Agent
NodeB
Total : 40c
Alloc : 38c
Usage : 10c
Agent

1.4. 借用的代价: 不稳定的生命周期

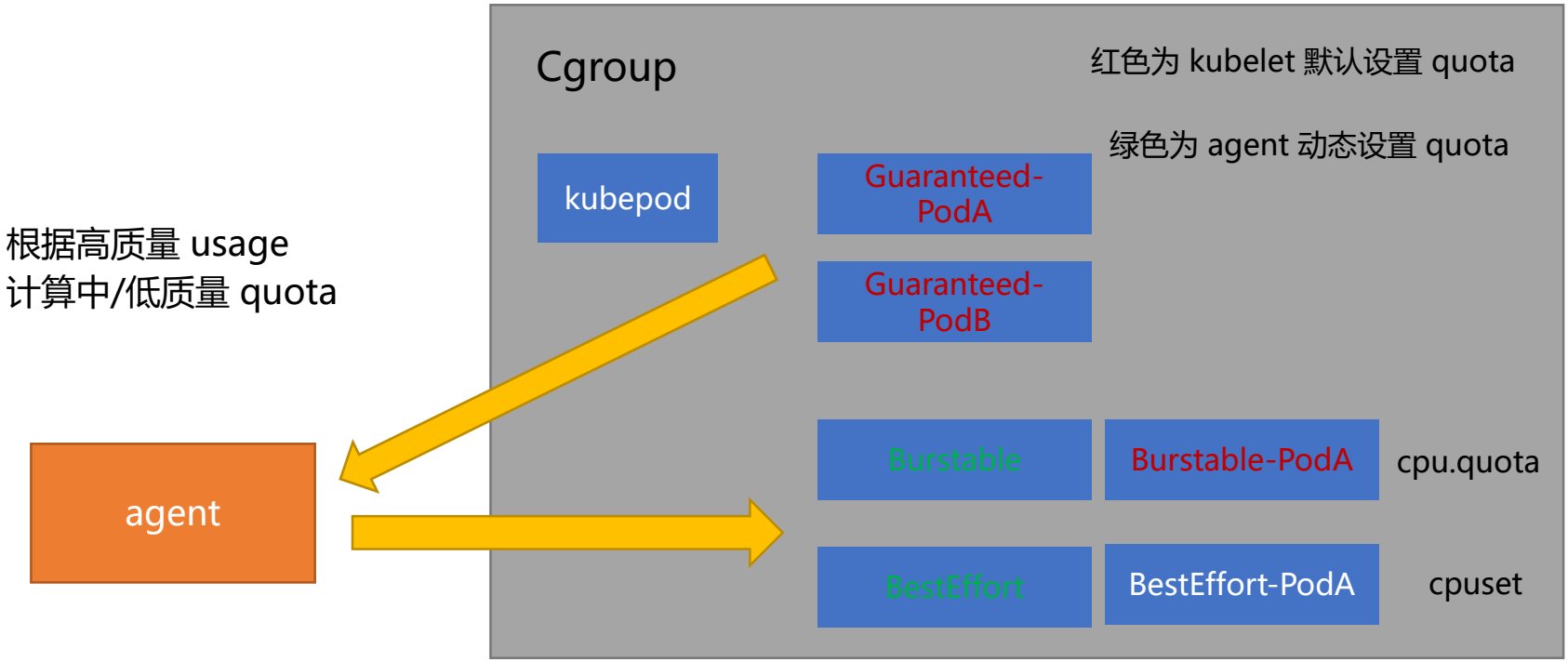
理想中的"借用”



现实中的"借用”

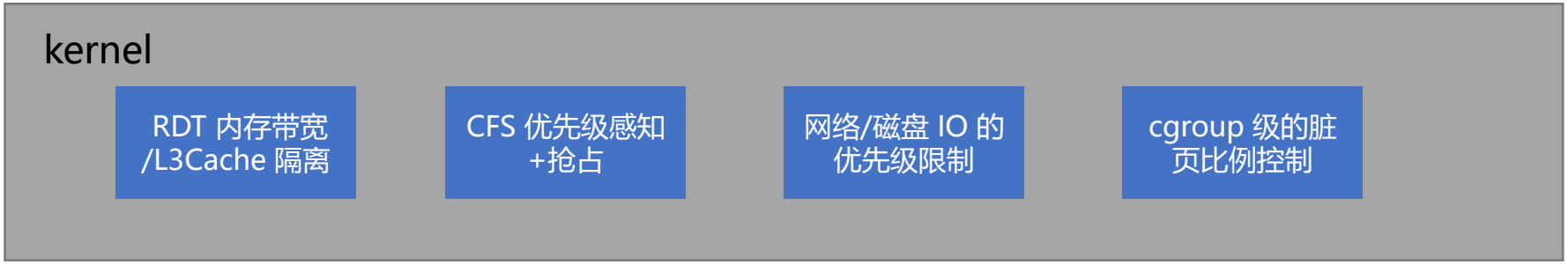


1.5. 单机引擎：隔离与退避

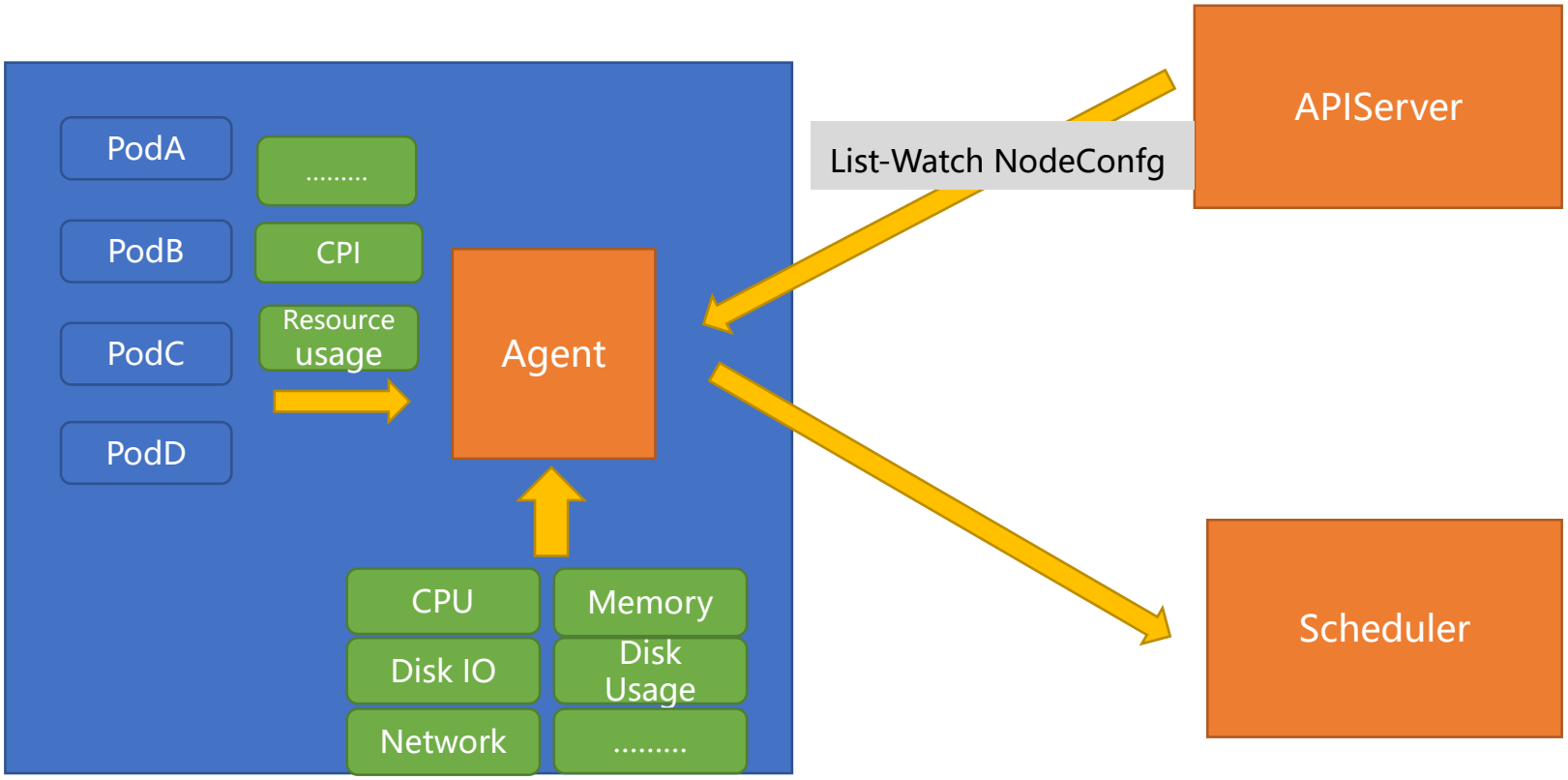


当 besteffort 框内用量小于 1c 时, 驱逐所有框内 Pod

当 guaranteed 用量持续上涨时, 降低 burstable 整体的 cpu.quota



1.6. 单机引擎：构建资源视图

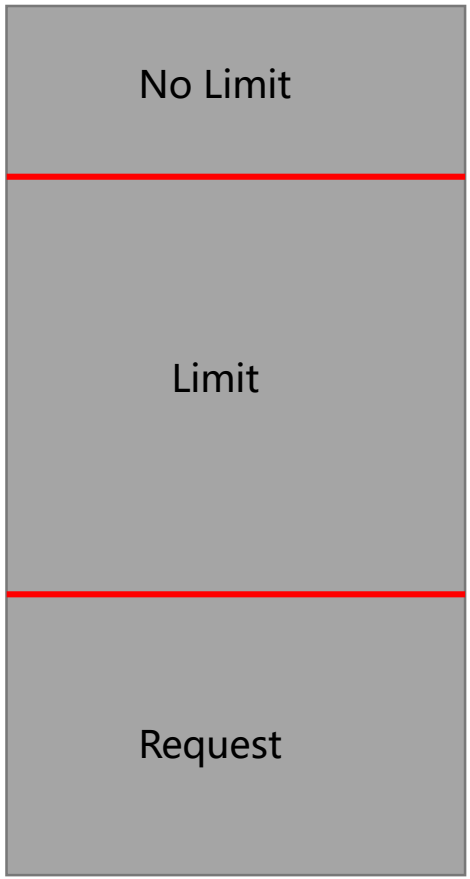


中等质量容器可用量 = 单机最大 CPU 用量 - 高质量容器用量 - Safety-Margin

低等质量容器可用量 = 单机最大 CPU 用量 - 高质量容器用量 - 中等质量容器用量 - Safety-Margin

1.7. 超发的结果: 质量分级

定价: A << B < C <D



besteffort → A

normal → B

stable → C

非混部 → D

1.8. 落地后遇到了哪些问题

- 热点问题：保证了用量，如何保证质量？

  热点迁移：出现热点，系统自动迁移容器

  热点预测：基于时间序列的热点预测，调度上避免热点

- Pending Pod：低质量需求激增带来的调度性能需求

  副本数托管：基于集群负载对应用进行动态扩缩容

  多调度器： 基于质量的多调度器 + 分片架构

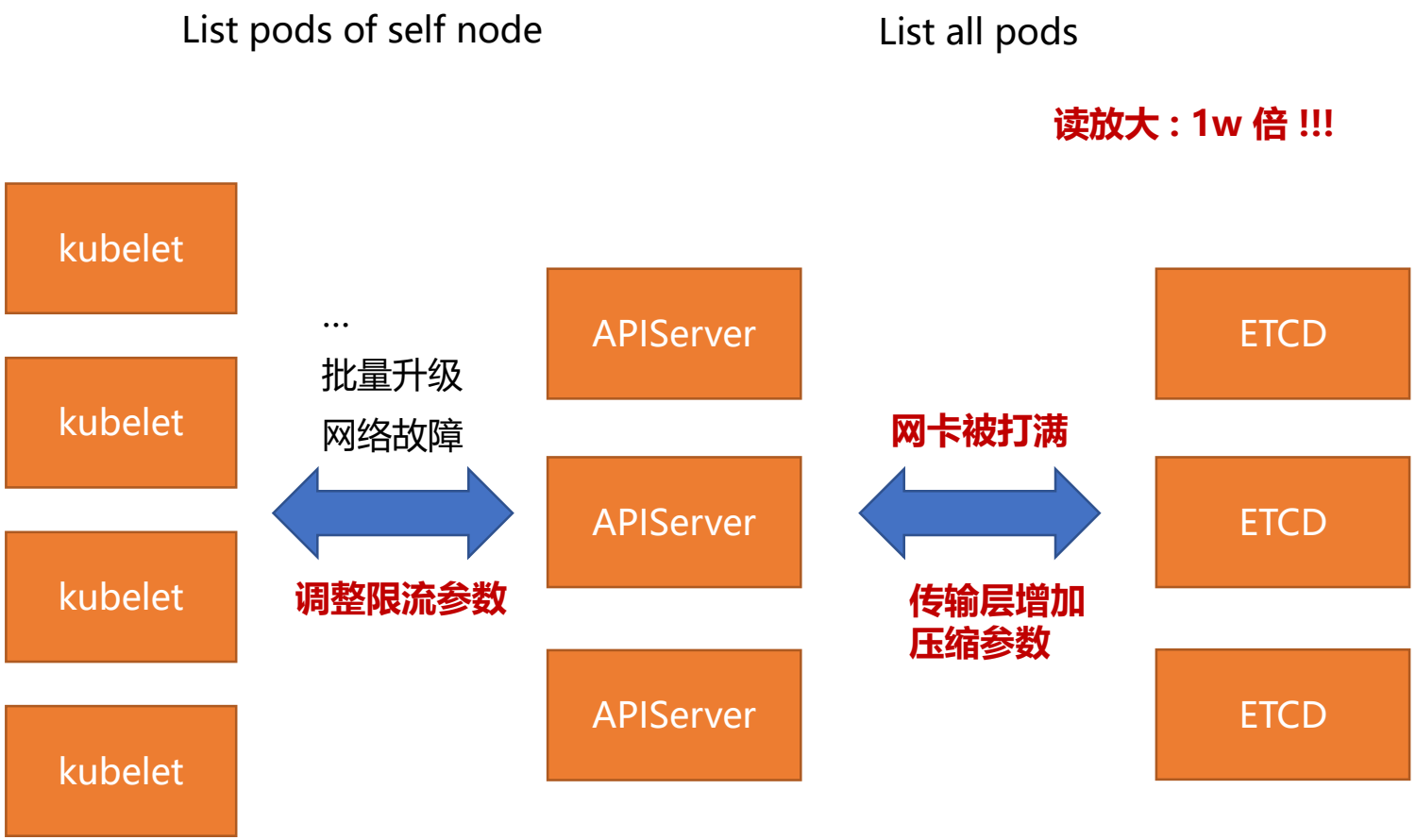
- 业务丰富度受限于机器数量

  扩大集群规模，产生性能问题

02

运维：大规模 kubernetes 集群建设

2.1. 从 case 来看 kubernetes 系统瓶颈



降级处理：让集群逐步消化掉流量

依然存在请求成功后很快又失败，重新 List-Watch 的情况

2.2. List-Watch 工作原理

List-Watch 工作原理

- List 请求获取全量数据, 同时通过版本号来保证一致性
- Watch 请求获取增量数据, 起始点使用 List 请求版本号

List 请求什么情况下请求会透传到 ETCD?

- ResourceVersion 为空
- Limit 不为 0 且 ResourceVersion 不为 0 : Informer 默认开启分页

```
pagingEnabled := utilfeature.DefaultFeatureGate.Enabled(features.APIListChunking)
hasContinuation := pagingEnabled && len(pred.Continue) > 0
hasLimit := pagingEnabled && pred.Limit > 0 && resourceVersion != "0"
if resourceVersion == "" || hasContinuation || hasLimit {
    // If resourceVersion is not specified, serve it from underlying
    // storage (for backward compatibility). If a continuation is
    // requested, serve it from the underlying storage as well.
    // Limits are only sent to storage when resourceVersion is non-zero
    // since the watch cache isn't able to perform continuations, and
    // limits are ignored when resource version is zero.
    return c.storage.List(ctx, key, resourceVersion, pred, listObj)
}
```

2.3. 从 case 来看 kubernetes 系统瓶颈

```go
func (kl *Kubelet) updateNodeStatus() error {
    klog.V( level: 5).Infof( format: "Updating node status")
    for i := 0; i < nodeStatusUpdateRetry; i++ {
        if err := kl.tryUpdateNodeStatus(i); err != nil {
            if i > 0 && kl.onRepeatedHeartbeatFailure != nil {
                kl.onRepeatedHeartbeatFailure()
            }
            klog.Errorf( format: "Error updating node status, will retry: %v", err)
        } else {
            return nil
        }
    }
    return fmt.Errorf( format: "update node status exceeds retry count")
}
```

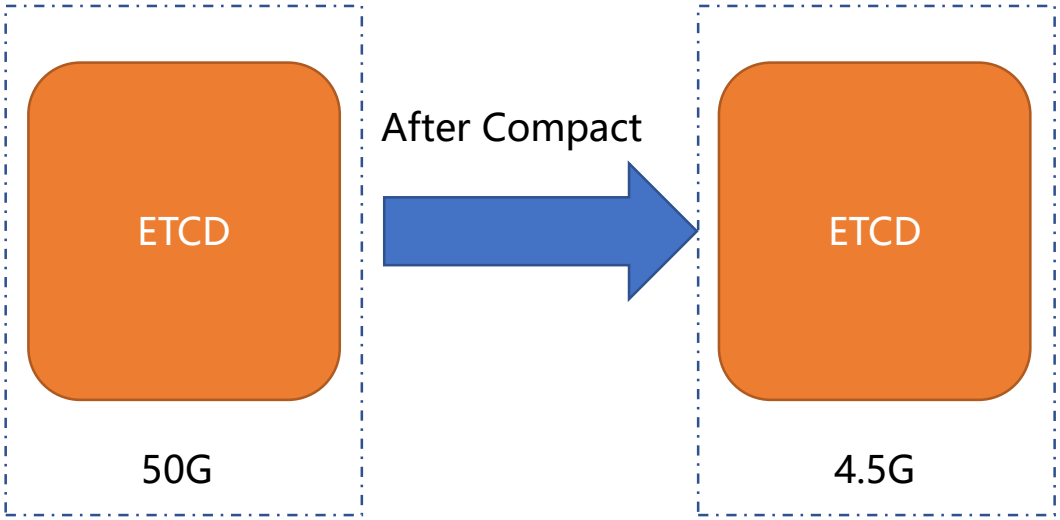
TryUpdateNodeStatus => Get Node => Update Node Status

onRepeatedHeartbeatFailure => close all connections

APIServer 限流参数 max-requests-inflight : 所有读请求

新增 APIServer 限流参数 max-requests-inflight-by-verb

2.4. 从 case 来看 kubernetes 系统瓶颈



压缩带来的问题：

1. 存储空间并没有真正释放, 被保存为 freelist 供二次使用
2. ETCD(boltDB) 在操作 freelist 时, 会带来额外的操作, 增加延迟
3. Compact 本身带锁, 会阻塞正常的读写请求

压缩时延迟飙高也会触发 kubelet 更新 NodeStatus 失败

2.5. 集群规模增大：哪些组件成为了瓶颈

ETCD	APISever	Controller	Scheduler	Operator
存储/一致性协议	流量/缓存	数据规模	数据规模	数据规模

集群规模扩大 => 资源数量增多

集群规模扩大 => 单机组件线性增长

无法通过运维手段解决性能瓶颈

单机 agent 影响范围急剧增大

kubelet	cni	hybrid	log	hardware
kube-proxy	csi	monitor	kernel	network

2.6. Agent 故障 : containerdc

```
"State": {
    "Status": "running",
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "Dead": false,
    "Pid": 4294967295,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2019-11-27T22:21:24.999965567Z",
    "FinishedAt": "0001-01-01T00:00:00Z"
},
```

https://github.com/containerd/containerd/pull/3857

带来的影响 : 整机进程全部**被 Kill** ！

总结与展望

质量分级

- 基于原生 Qos 构建了质量体系
- 未来质量体系纳入更多维度的资源

单机压制/隔离

- 可压缩资源的动态调整, 按照 quota 等比隔离资源
- 未来基于 ebpf 建立更细粒度的资源观测体系

大规模 kubernetes 集群建设

- 增强 cache 功能, 绝大部分场景不需要穿透至 ETCD
- 未来持续优化 ETCD 写性能

Q & A

THANKS

麦思博(msup)有限公司是一家面向技术型企业的培训咨询机构，携手2000余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过3000余家企业续约学习，是科技领域占有率第1的客座导师品牌，msup以整合全球领先经验实践为己任，为中国产业快速发展提供智库。

高可用架构公众号主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。
高可用架构系列社群是一个社区组织，其精神是“分享+交流”，提倡社区的人人参与，同时从社区获得高质量的内容。