Cloud Native Summit 专场

本期议题：字节跳动基于 Kubernetes 的大规模集群联邦技术实践

刘晟丽 2023年05月28日

目录

1 背景

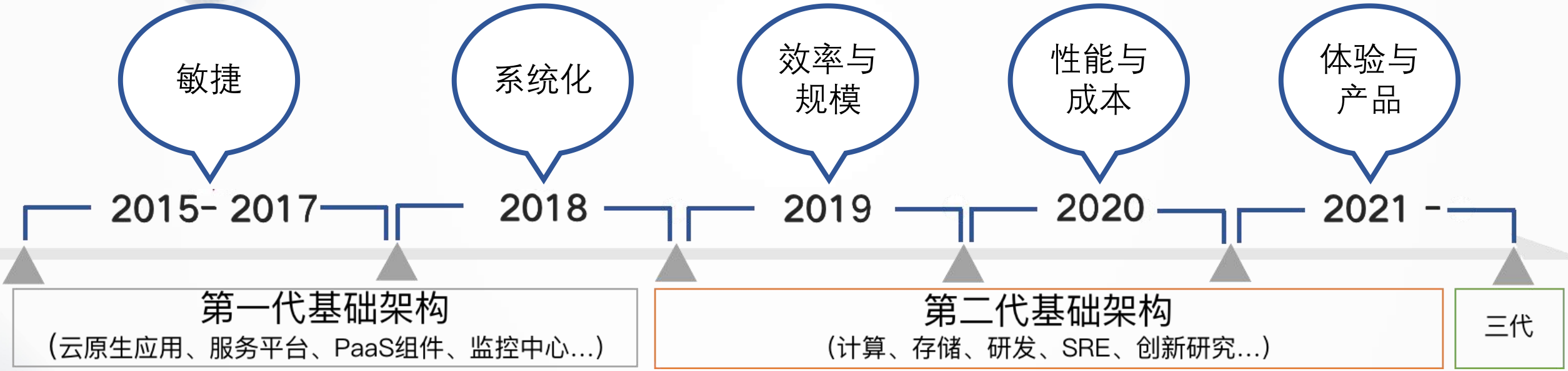
2 2019：基于 KubeFed 的集群联邦

3 2021：第二代联邦 KubeAdmiral

4 当下与未来

背景

字节跳动云原生发展历程



第一代基础架构

- 核心业务服务云原生化
- TCE 平台建设完成
- 统一的公司级 SRE 体系

第二代基础架构

- 关注集群规模与资源效率
- 2019 年以 KubeFed 为基础引入集群联邦

第三代基础架构

- 应用多样化，调度需求精细化
- 2021 年第二代联邦 KubeAdmiral

字节跳动大规模集群联邦

GOTC

资源体量

联邦总节点数：21w+
在线微服务：10w+
Pod: 1000w+

发布效率

TCE 日变更数：2w+
发布过程平滑、高效

支撑亿级DAU的工程实践

业务类型

在线微服务
延迟敏感型 socket 服务
有状态服务

容灾要求

多VAU容灾
集群级别 rebalance 容灾

2019 以 KubeFed 为基础引入集群联邦

管理众多 k8s 集群面临的问题

用户心智负担重

随着 K8s 集群规模逐步扩大，物理集群数量达到 500+, 给用户在集群选择时带来额外的心智负担。对于实例数量超过单个集群可用资源时，需要在新的集群上创建同名服务

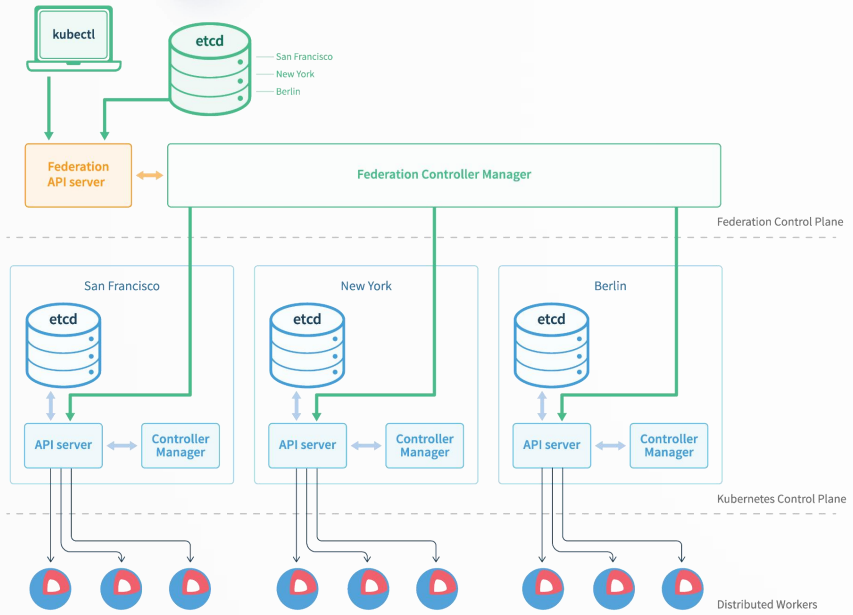
集群运维复杂

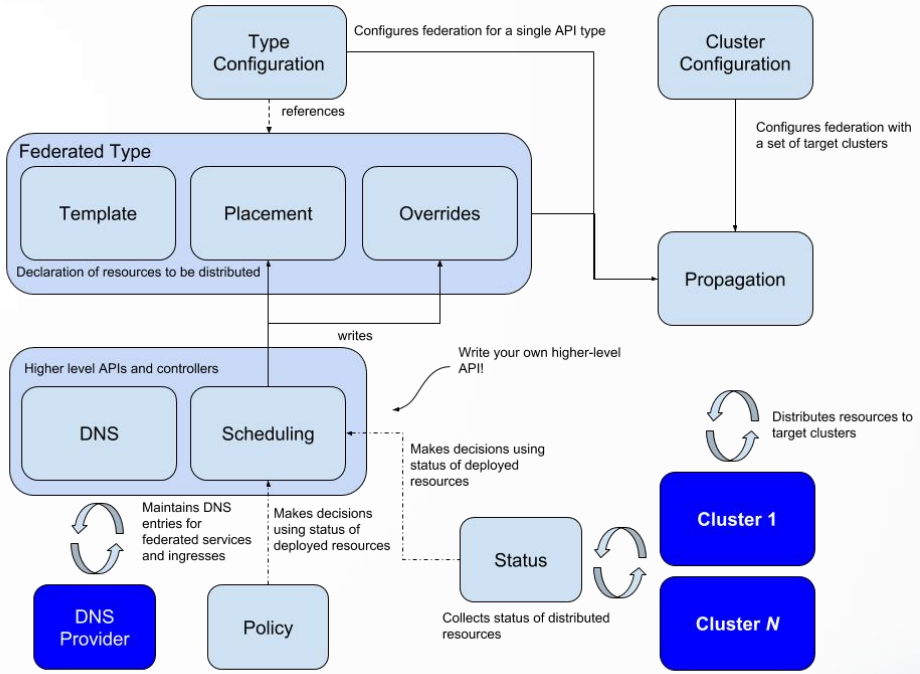
繁多的集群促成了集群与特定的业务线绑定的局面，使得SRE日常运维操作需要与业务方协作完成

资源孤岛

集群间的部署率差异较大，空闲的资源总是分散在各个集群中，经常需要在集群间拆借机器，非常低效

联邦方案的选择：Federation v1 vs Federation v2





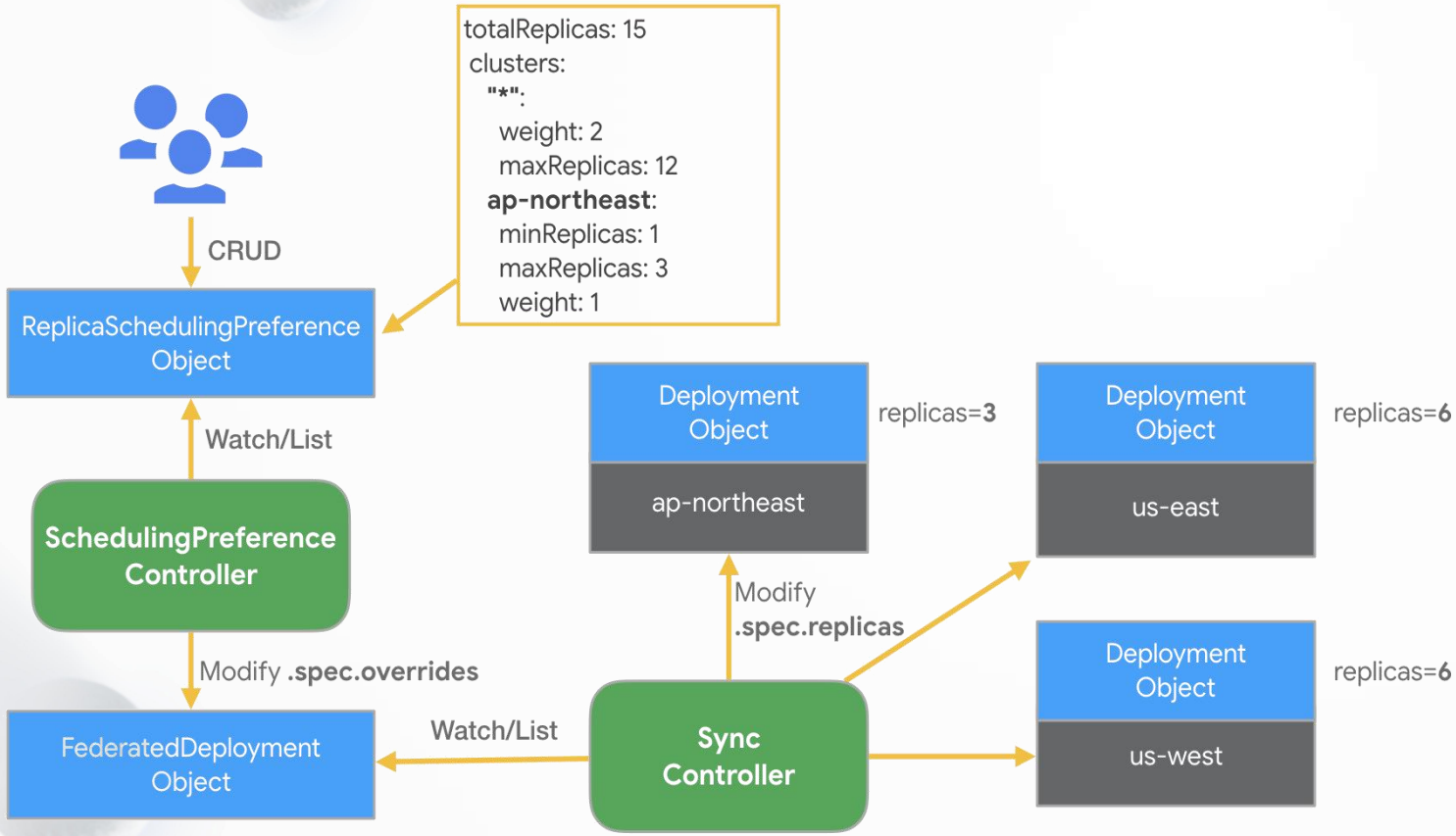
Federation v1:

- 只支持特定版本的 API resource, 扩展性差。
- 不支持 CRD
- 缺乏联邦调度策略

Federation v2:

- 扩展性: 引入 FederatedType, 通过 Template、Placement、Overrides 三个字段能够方便地将任意对象“联邦化”。
- 调度语义: 支持 ReplicaSchedulingPreference, 能根据静态权重调度 deployment 副本数。

以 KubeFed (Federation v2) 为基础的集群联邦




```yaml
kind: FederatedDeployment
spec:
  overrides:
  - clusterName: ap-northeast
    clusterOverrides:
    - path: /spec/replicas
      value: 3
  - clusterName: us-east
    clusterOverrides:
    - path: /spec/replicas
      value: 6
  - clusterName: us-west
    clusterOverrides:
    - path: /spec/replicas
      value: 6
  placement:
    clusters:
    - name: ap-northeast
    - name: us-east
    - name: us-west
  template:
    ...
```

KubeFed 无法满足生产环境的要求

规模性问题

联邦控制面分 DC 部署，
单个联邦控制面规模庞大:
node: 7w+
deployment: 10w+
pod: 1000w+
导致 controller manager 内存爆炸, sync 延迟增长

资源利用率

每个 member 集群都通过静态的 weight 值定义权重，导致不同 member 集群部署水位不均，资源利用率低

用户体验

相比单机群 kubernenets，服务扩缩容过程不够平滑，经常发生实例“迁移”现象

规模性问题：Controller 分片

问题

- 线上单个集群最大规模：
  - 1.8w node, 80 w pod
- 单个联邦控制面规模:
  - node: 7w+
  - deployment: 10w+
  - pod: 1000w+
- 由此引发的性能问题：
  - controller 内存爆炸，联邦系统可伸缩性差
  - controller sync 延迟升高
  - controller 重启等待时间过长，导致 apiserver 内的 watch cache buffer 过期

规模性问题：Controller 分片

解法

- 对数据进行分片，每个controller只需要关注分片内部的对象，即可极大地降低内存占用和 sync 延迟

- 通过 ListOption 支持 ShardingCountCount/ ShardingIndex/ ShardingLabelKey/ HashFN 等参数，在 apiserver 侧根据分片规则进行过滤。通过并行，预期 List 耗时从分钟级降低到秒级

- Watch 分片会打破严格全局有序的规则，但是 Kubernetes 本质是一个最终一致性模型，不同对象之间的 Event 顺序没有要求，因此只要保证单个对象的 Event 有序即可，不同分片之间的时间允许乱序

```
HashFunc(ShardingLabelKey) % ShardingCount == ShardingIndex
```

资源利用率：基于集群水位的动态权重调度

- KubeFed 的副本调度策略 RSP 只能为每个 member 集群设置静态权重：
  - 静态权重很难做到集群部署率一致，在个别集群部署率很高时，很容易产生pending。在实际生产实践中，我们期望在SRE做好集群规划后，调度器能自适应的向各个子集群中调度实例

- 引入基于集群水位的动态权重调度:
  - 进行权重计算时，同时考虑集群的资源总量 allocatable 和当前可用资源量 available

$$R_i = Min(A_i/Sum(A), 1.4 * T_i/Sum(T)) \quad \#\ 1.4\ 为\ SRE\ 最佳经验比例$$

After normalization: $r_i = R_i \ / \ Sum(R) \ * \ R_{total}$

- 最终，所有 member 集群的部署率都维持在 95% 以上，部分大规模 member 集群部署率可以达到 98%

用户体验：扩缩容过程中避免实例迁移

问题

- 30 个实例分布在 ABC 3个 member 集群，现在用户想缩容到 15 个实例：
  - 用户预期：停止 15 个实例
  - 按权重调度：停止 20 个实例，启动 5 个实例

集群	A	B	C
权重	10	10	10
实例数	15	15	0

缩容：30 -> 15

集群	A	B	C
权重	10	10	10
实例数	5	5	5

- 30 个实例分布在 ABC 3个 member 集群，现在用户想缩容到 15 个实例：
  - 用户预期：停止 15 个实例
  - 按权重调度：停止 20 个实例，启动 5 个实例

集群	A	B	C
权重	10	10	10
实例数	15	15	0

扩容：30 -> 36

集群	A	B	C
权重	10	10	10
实例数	12	12	12

用户体验：扩缩容过程中避免实例迁移

解法

- 目标：在实例不发生迁移的条件下，尽量让结果趋近于权重分布
- 改进算法：以当前分布和预期分布的差值，作为权重进行调节

集群	A	B	C
权重	10	10	10
实例数	15	15	0

缩容：30 -> 15

集群	A	B	C
权重	10	10	10
实例数	8	7	0

1. current distribution = [15, 15, 0], total replicas: 30
2. disired distribution = [5, 5, 5], total replicas: 15
3. distance = disired - current = [-10, -10, 5], total distance: 15
4. 对于缩容场景，去掉正数项 distance = [-10, -10, 0]
5. 以 distance 为权重，重新分配差值 15：[-7, -8, 0]
6. 最终调度结果：[-7, -8, 0] + [15, 15, 0] -> [8, 7, 0]

2021 第二代联邦 KubeAdmiral

Kubefed V2 面临的问题

接入难度高

由于联邦层暴露的是“联邦化”的对象，而不是 Kubernetes 原生对象，上层服务往往需要针对联邦集群做特殊适配

调度扩展性差

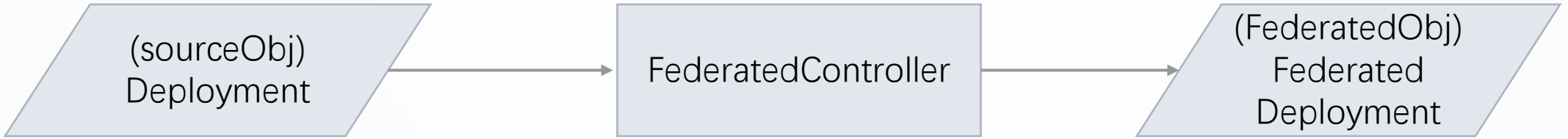
只支持 RSP 副本调度，对其他调度策略只能在程序中特殊处理，扩展性差

业务稳定性

调度过程中若发生 Rebalance，可能会影响业务稳定性；同时，升级过程中容易发生子集群进度不一致，极大影响发布效率

接入难度高：支持原生 API

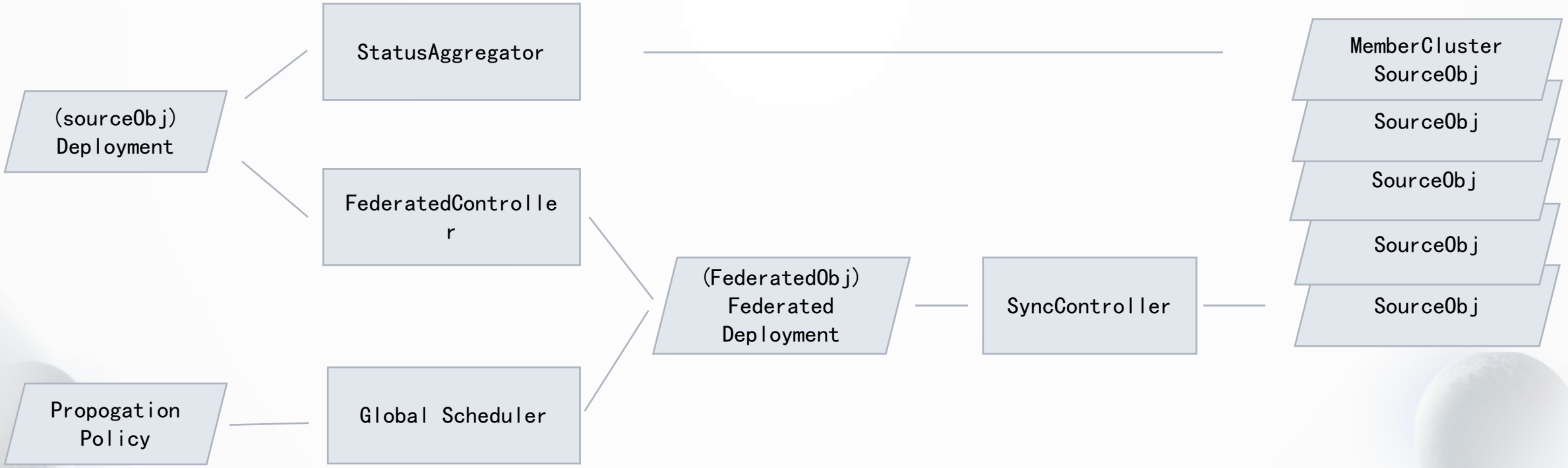
引入 FederatedController，负责将原生的资源转换为 FederatedObject。




```
// FederatedObject
{
  metav1.TypeMeta
  metav1.ObjectMeta
  Spec
      GenericPlacementSpec
      GenericOverrideSpec
      Template
      RevisionHistoryLimit
      RetainReplicas
  Status
}
```

接入难度高：支持原生 API

通过引入 StatusAggregator，负责将子集群资源的状态聚合到原生资源，针对不同的资源可以添加聚合 Plugin。

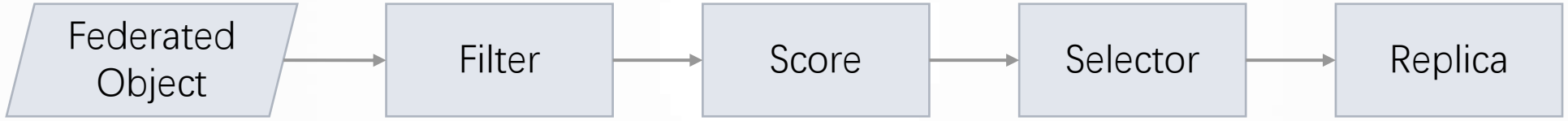


调度扩展性差：Global Scheduler

参考 Kubernetes Scheduling Framework，建设面向多样化 FederatedObject 的调度器：Global Scheduler。


```
// FederatedObject
{
  metav1.TypeMeta
  metav1.ObjectMeta
  Spec
      GenericPlacementSpec
      GenericOverrideSpec
      Template
      RevisionHistoryLimit
      RetainReplicas
  Status
}
```

调度扩展性差：Global Scheduler



Global Scheduler 的 Framework 和 Plugin：

- Filter：根据子集群的状态、亲和性、资源、是否支持 CRD 等等过滤不符合条件的 Member Cluster
- Score：对符合条件的 Members Cluster 打分，例如资源、亲和性等
- Selector：最多分发到多少个集群
- Replica：面向类似 Deployment 提供的副本数调度

调度扩展性差：Global Scheduler

Global Scheduler 支持丰富的调度策略模板，用户可在应用中指定所需要的调度策略模板

```
type PropagationPolicySpec struct {
    SchedulingProfile                 // 灵活的指定 Scheduling Profile
    SchedulingMode                    // 是否为副本数调度
    StickyCluster                     // 仅在首次调度，适合有状态服务
    ClusterSelector                   // 类似 NodeSelector
    ClusterAffinity                   // 类似 NodeAffintiy
    Tolerations                       // 污点与容忍配置
    MaxClusters                       // 最多可分发到多少个子集群
    Placements                        // 用户指定集群或者静态权重
    DisableFollowerScheduling         // 是否开启依赖调度
}
```

业务稳定性：升级时控制子集群间的滚动粒度

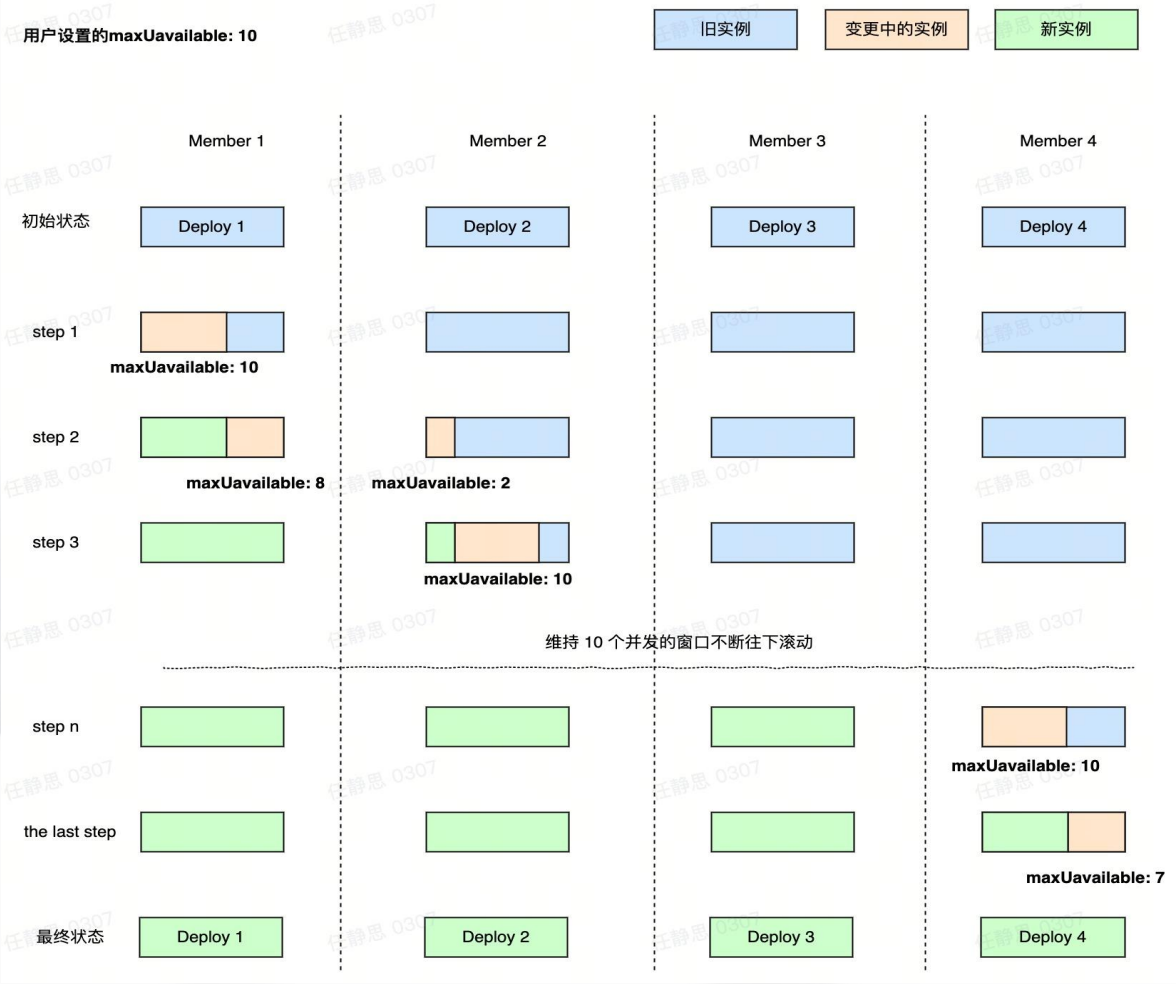
问题

- 稳定性问题
  - 发生 Rebalance 时，当前实例分布和调度结果可能有很大差异，容易发生实例不可用的数量超出预期
  - 对于实例数较小的服务，假如分布在多个子集群上，升级时实际滚动粒度很可能偏大
- 效率问题
  - 服务升级尾声时，由于子集群之间发布进度有差别，实际滚动变慢

业务稳定性：升级时控制子集群间的滚动粒度

解法

- 升级在子集群间逐个进行，根据实时状态控制窗口滑动，保持实际滚动粒度始终和业务配置一致



$$
\begin{aligned}
&= \sum available_i - MinAvailable \\
&= \sum available_i - (D - MaxUnavailable) \\
&= MaxUnavailable - D + \sum available_i \\
&= MaxUnavailable - (D - \sum r_i) - \sum (r_i - available_i) \\
&= MaxUnavailable - (\sum_{missing} d_i + \sum (d_i - r_i)) - \sum (r_i - available_i)
\end{aligned}
$$

当下与未来

KubeAdmiral 演进 —— 算力调度

作为应用的统一入口，需要从全局的视角，需要回答应用、机器、机房网络拓扑，这三者之间优化匹配空间如何。以算力为例，随着硬件平台的推陈出新和每年大量采购沉淀，公司已积累丰富多样的 CPU 型号，不同型号或存在单核巨大的算力差异，给业务带来负载不均等痛点。

- 做好算力的评估压测，得出不同服务在不同代际的 CPU 的性能体现。
- 做好算力的调度，为不同的服务分配有效收益更高、性能更一致的算力，从而优化算力的有效性。

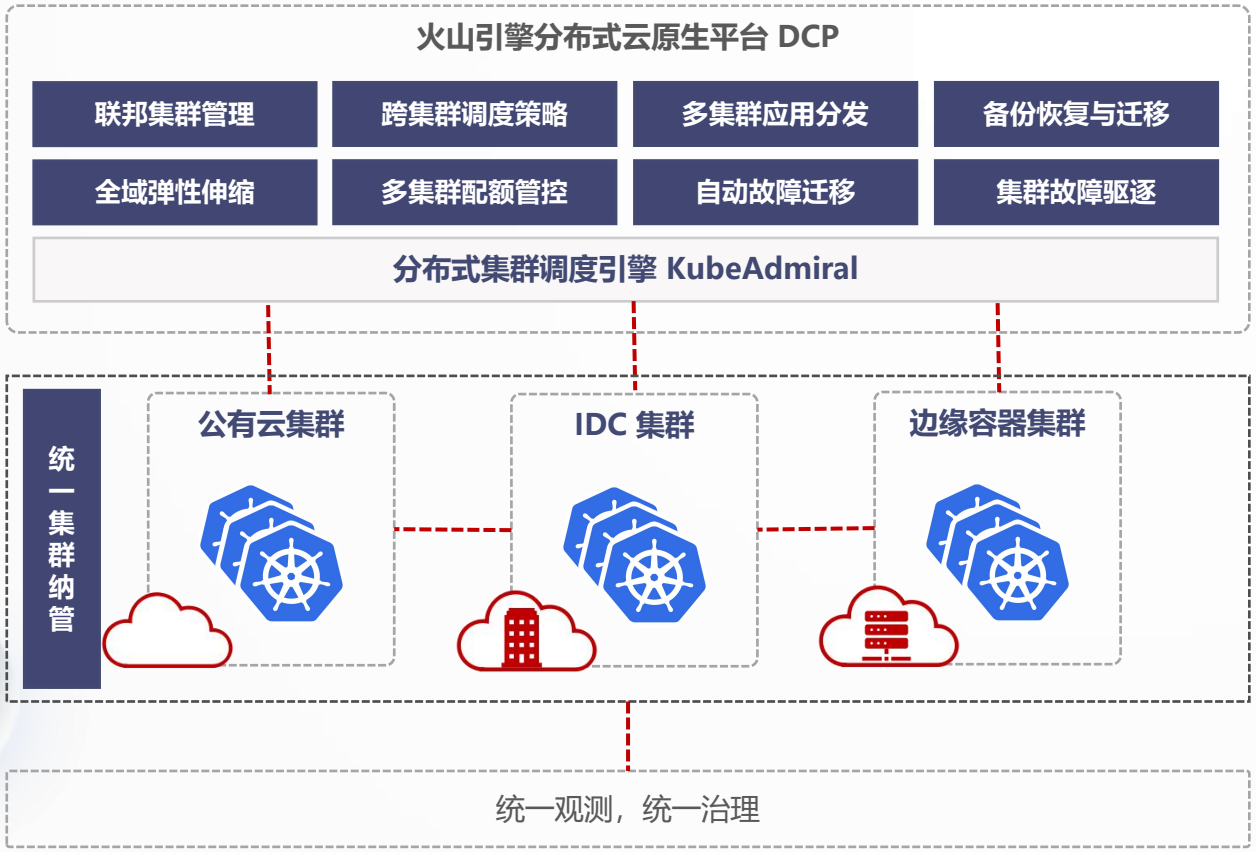
0.74	0.78	0.75	N/A	N/A	0.85	0.86	1	0.87	0.97
0.79	0.73	0.69	0.77	N/A	0.92	0.94	1	0.9	N/A
0.71	0.56	0.61	0.83	N/A	0.75	0.77	1	0.65	0.69
0.9	0.74	0.73	N/A	N/A	0.95	0.95	1	0.9	N/A
0.91	0.74	0.74	N/A	N/A	0.93	0.95	1	0.87	N/A
0.88	0.83	0.81	N/A	N/A	0.89	0.91	1	0.99	N/A
N/A	0.85	0.75	0.85	0.72	N/A	0.98	1	0.91	0.88
0.72	0.68	0.67	N/A	N/A	0.82	0.85	1	0.83	N/A
0.75	0.66	0.68	0.87	N/A	0.86	0.83	1	0.76	0.75
0.76	0.7	0.81	N/A	N/A	0.87	0.88	1	0.9	N/A
N/A	0.82	0.77	0.85	0.78	0.98	0.94	1	0.91	0.89
0.68	0.69	0.7	N/A	N/A	0.92	0.89	1	0.85	N/A
N/A	0.84	0.83	0.92	0.89	0.95	0.95	1	0.93	0.9
0.76	0.7	0.7	N/A	N/A	0.87	0.88	1	0.76	N/A

KubeAdmiral 演进 —— 多层调度协同

本质要回答，多集群的调度如何和单集群的调度做好协同，具体包括：

- 在机制上：如何和单集群的做好反馈机制，提升调度精确性，同时需对齐策略和数据？
- 在并池上：针对不同的 Workload，如何做好协同，比如调度，优先级，准入驱逐等？
- 在仿真上：如何构建整体的仿真能力，指导优化的方向？

KubeAdmiral 演进 —— 企业级云原生管理新模式



字节多云架构大规模实践

- 超大规模：管理多集群规模超过 21 万节点、1600 万核的资源
- 用户体验好：无需感知底层基础设施资源
- 容灾能力强：提供一键迁移
- 成本优化：字节内部部署率高达 97% - 98%

云原生应用管理新模式

- 统一管理平面
- 统一集群运维
- 统一算力分发
- 统一流量管控

KubeAdmiral 演进 —— OpenSource

KubeAdmiral 已经开源：https://github.com/kubewharf/kubeadmiral

之后将吸取社区经验与反馈，并不断更新和完善KubeAdmiral的功能，以更好地回馈社区。

扫码关注字节跳动云原生

扫码加入字节跳动云原生社群

GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE