KUBERNETES COMMUNITY DAYS BEIJING 2025

基于控制面单元化的 Kubernetes 集群联邦

任静思 字节跳动资深研发工程师

Content

目录

Part 01

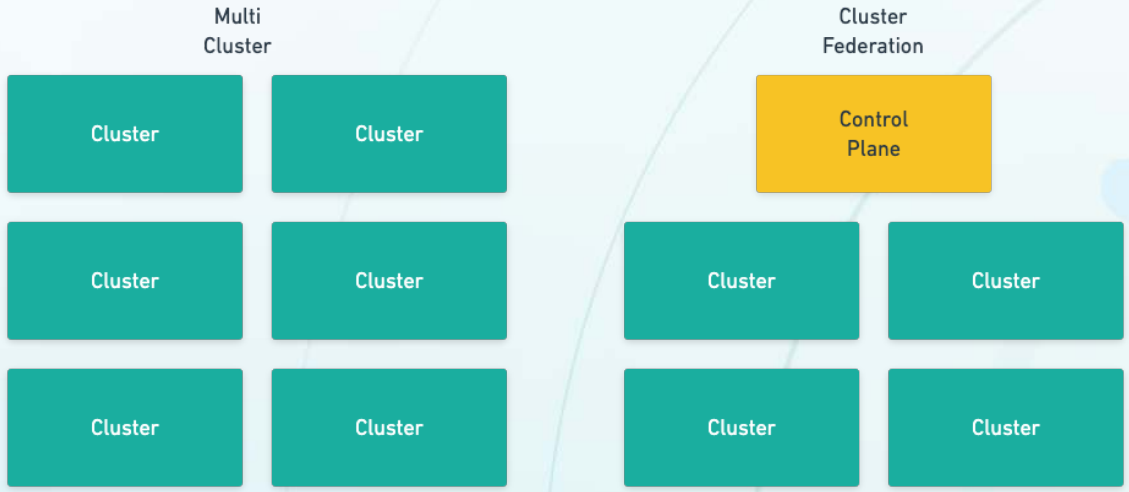
背景

大规模集群联邦带来的性能与稳定性挑战

大规模生产环境集群联邦

- 在字节跳动，我们采用 Kubernetes 集群联邦作为全局资源的统一入口
    1. 资源体量
        1. 联邦总节点数：21 W+
        2. 在线微服务数：10 W+
        3. Pod: 1000 W+
    2. 业务类型
        1. 在线微服务，延迟敏感 Socket 服务，有状态服务，批处理任务
    3. 发布效率
        1. 日变更数：3 W+，发布过程需要平滑，高效
    4. 容灾要求
        1. 多 VAU 容灾
        2. 集群级别 rebalance 容灾

大规模集群联邦带来的性能与稳定性挑战



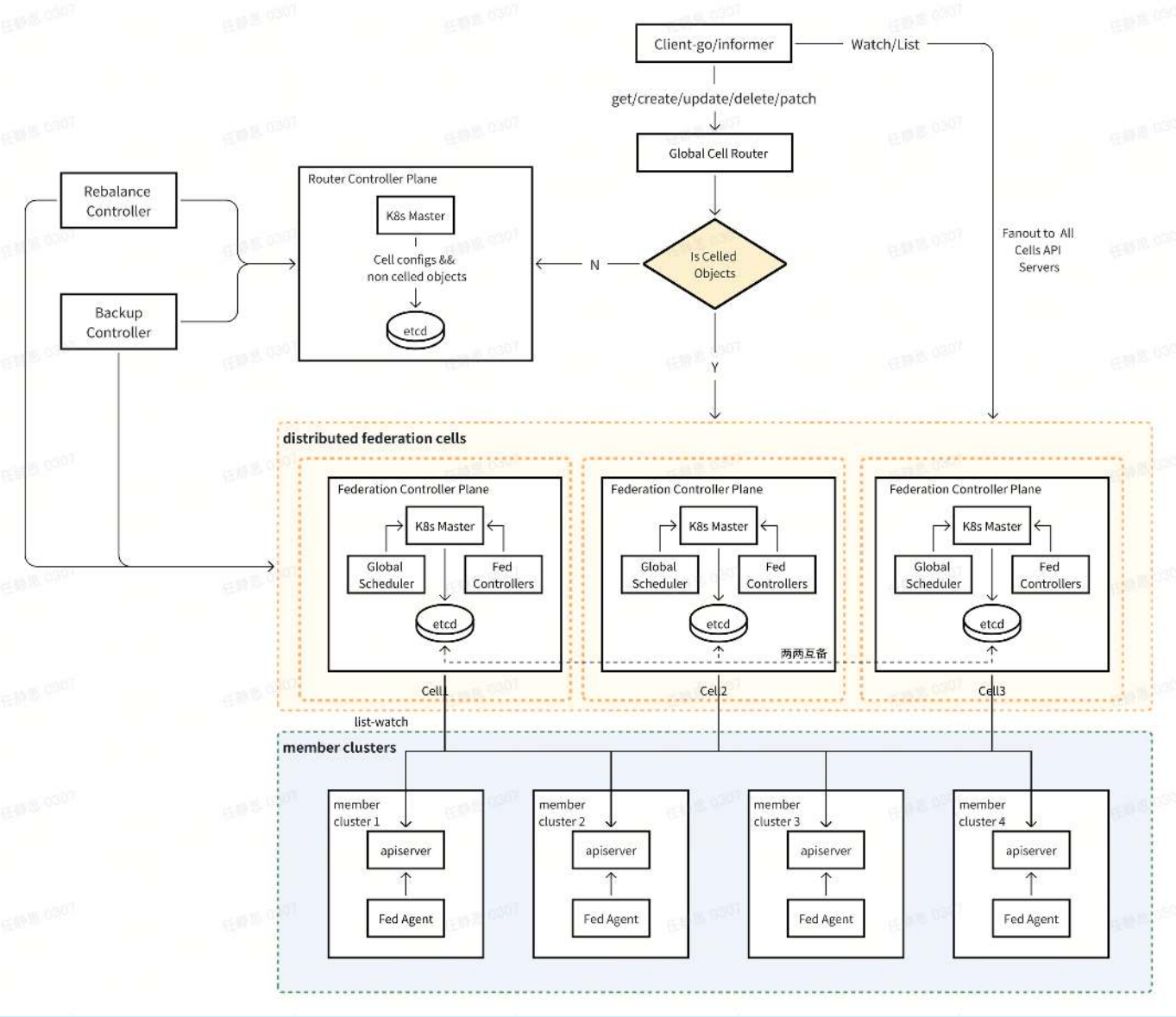
- 联邦控制面本身也是一个中心化系统：
  - 系统可扩展性受限：单个 Kubernetes 集群作为联邦控制面，承载能力有上限，当业务规模达到阈值，需要考虑通过水平拆分多个控制面集群，提高系统的线性扩展能力。
  - 缺乏高可用性：一旦联邦控制面发生故障会导致整个集群池化资源不可用，风险较大。通过拆分多个联邦控制面的方式，一方面可以减小并隔离故障域，另一方面引入控制面之间的相互备份，在灾难场景下做到快速恢复。

Part 02

基于控制面单元化的 Kubernetes 集群联邦

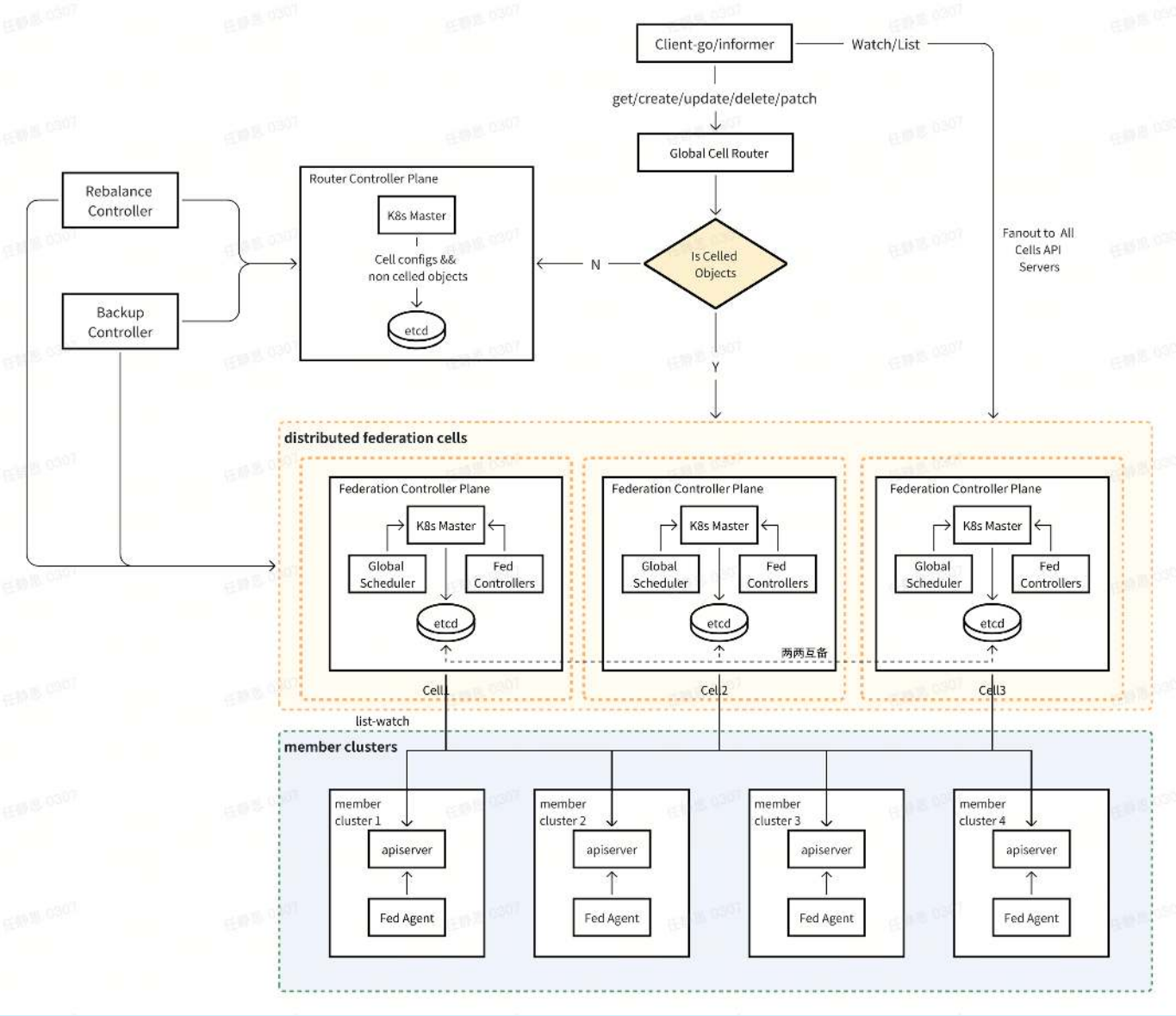
大规模集群联邦带来的性能与稳定性挑战

总体设计



- 控制面单元化：
  - 联邦控制面拆分成若干个独立的单元，每个单元独立部署 etcd, apiserver 和 fed controller
  - Workload 通过负载均衡算法被路由到某一个控制面单元中，在该单元中完成子集群调度，传播，状态收集等能力

总体设计



- 控制面单元化：
  - Global Cell Router: 负责根据映射算法，把 Workload 映射到某个单元（Cell）中; 对外提供单个对象的 k8s API, 单元化架构对上层 client 透明
  - Router Control Plane: 存储单元化配置和不需要单元化的对象
  - Rebalance Controller: 根据映射算法，对 Workload 在单元间迁移再平衡
  - Backup Controller: 负责 Workload 在单元间的备份

Part 03

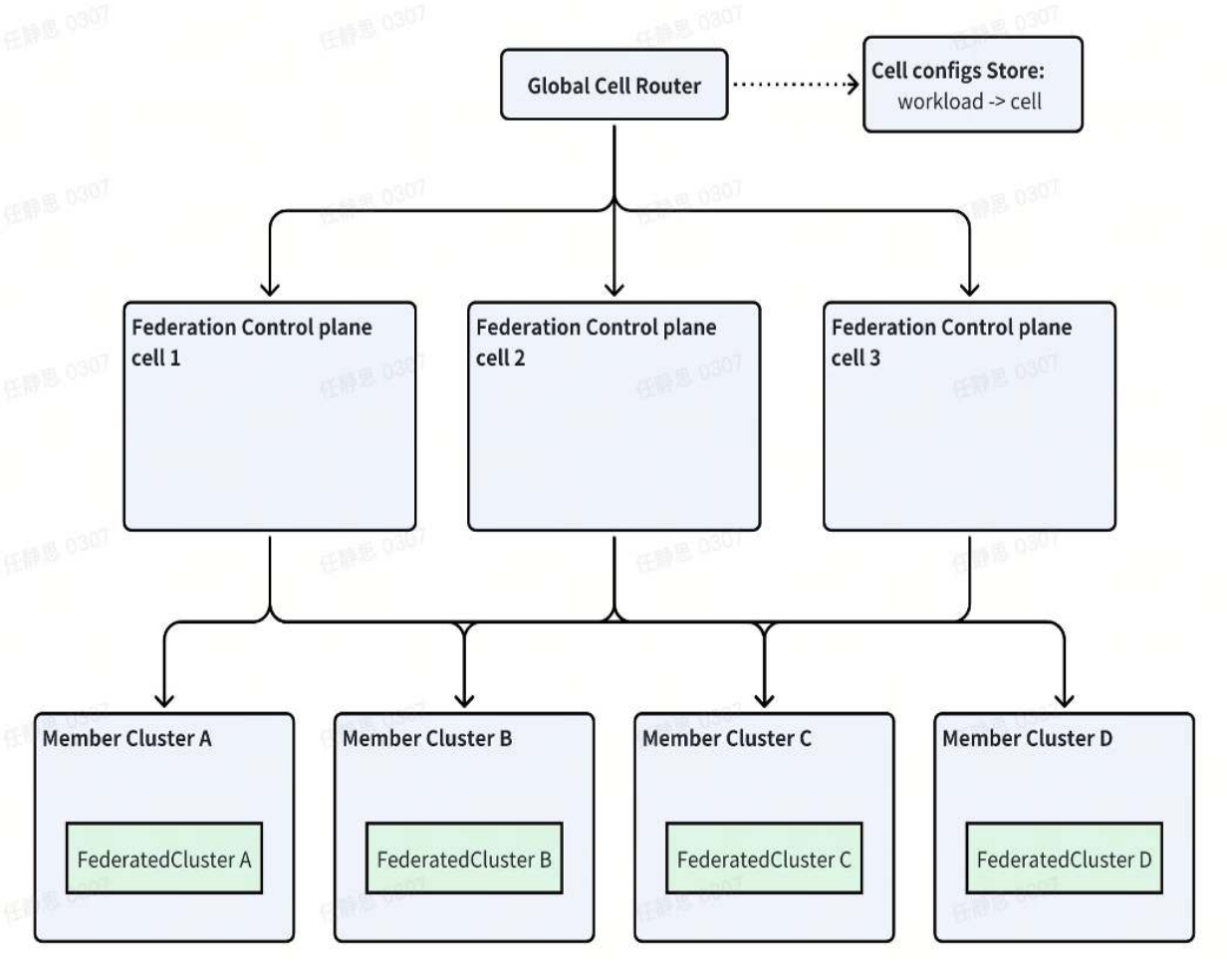
关键技术细节

大规模集群联邦带来的性能与稳定性挑战

映射算法

- 映射算法解决 Workload 具体被路由到哪个单元的问题，Workload 随着应用发布动态创建，其生命周期受对应单元的 Fed controller 管理
  - 负载均衡：各个单元间的 Workload 的数量应该相对均衡，与每个单元的承载能力成正比
  - 去除单点瓶颈：Workload 与单元间的映射关系不应该依赖于某种单点的存储系统记录，最好能够根据配置和算法进行计算
  - 增添/裁撤单元：随着联邦系统的容量扩展和升级，会出现增添新单元以及裁撤旧单元的需求，此时 Workload 和单元之间的映射关系会发生改变，存在 Workload 在单元间迁移的过程，这一过程最好是对外透明的，只影响正在迁移的 Workload
  - 容灾备份：Workload 常态下存储在主单元中，受主单元的 Fed controller 管理，一旦主单元发生不可用故障，需要能够快速切换到备份单元中，由备份单元的 Fed controller 接管

映射算法



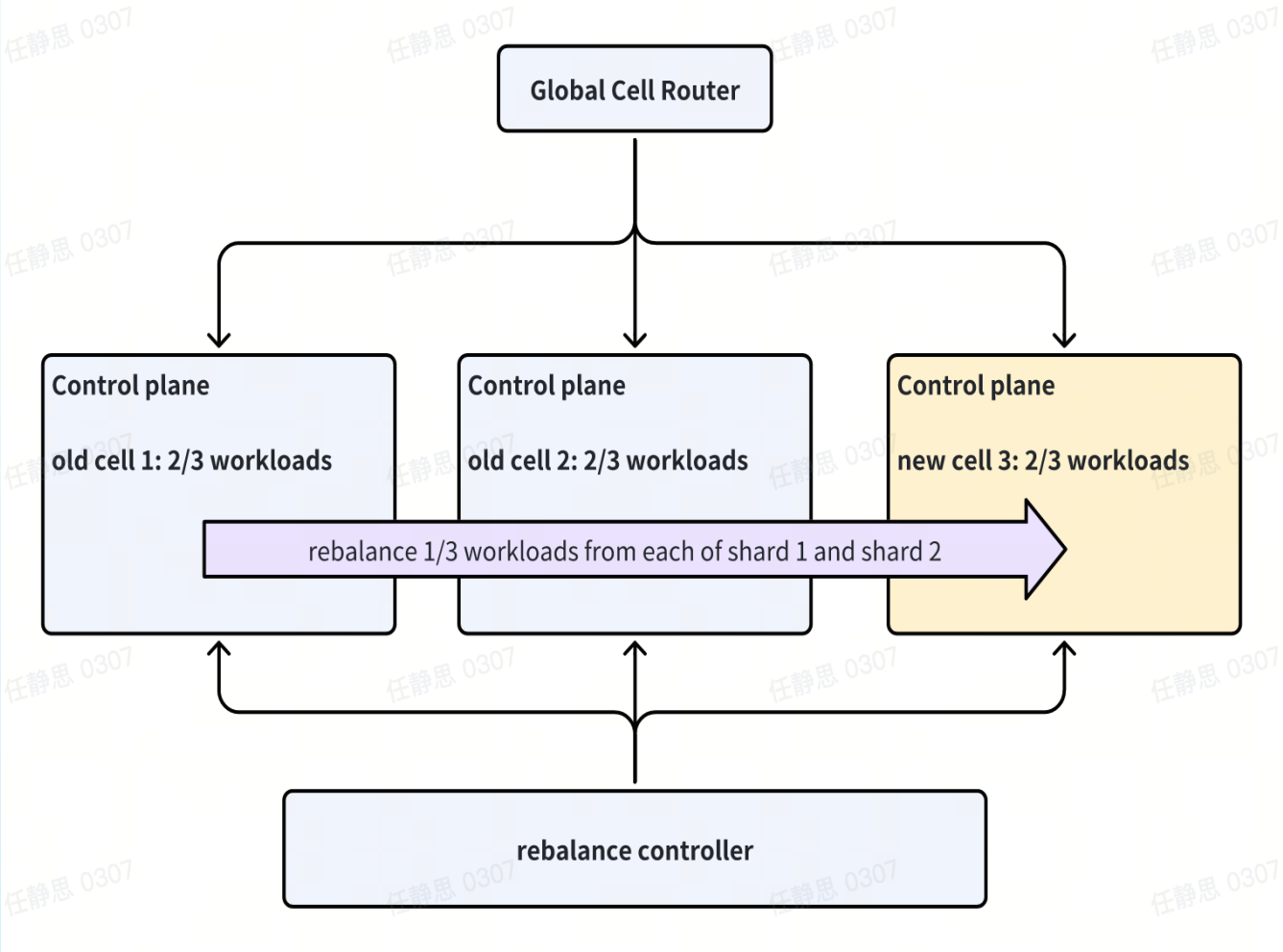
- 哈希算法：

ShardID = Hash(name) mod ShardNumber

- 一致性哈希算法：

ShardID = ConsistentHash(name, ShardNumber)

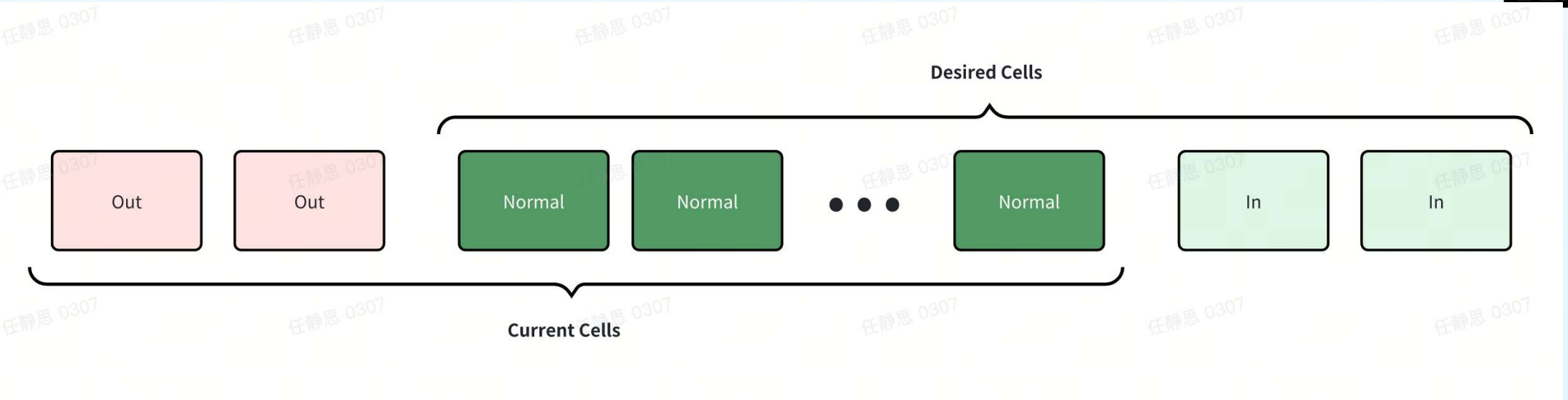
增加/裁撤单元



设计映射算法，配合 rebalance controller:

- 移动尽量少的 Workload, 使单元间重新达到负载均衡
- Workload 的迁移过程需要是有序可控的
- Workload 迁移期间，映射算法能够正常工作，把请求转发到对应的单元
- 不依赖中心化的存储

增加/裁撤单元



在单元扩缩容完成前，单元集合：

$$CurrentCells = OutCells \cup NormalCells$$

单元扩所容完成后，单元集合：

$$DesiredCells = InCells \cup NormalCells$$

增加/裁撤单元

- 对于正在进行迁移的 workload, 将其 rebalanceStatus 标记为 migrating
- 在单元扩缩容期间，router 对每个 workload 请求，会调用两次一致性 hash 函数：
- 上述两个 CellID 可能有四种组合

$$OldCellID = ConsistentHash(name, CurrentCells)$$
$$NewCellID = ConsistentHash(name, DesiredCells)$$

	$OldCellID \in OutCells$	$OldCellID \in NormalCells$
$NewCellID \in InCells$	情况一：workload 需要从 out 单元迁移到 in 单元	情况二：workload 需要从 normal 单元迁移到 in 单元
$NewCellID \in NormalCells$	情况三：workload 需要从 out 单元迁移到 normal 单元	情况四：workload 保持在 normal 单元中

增加/裁撤单元

	$OldCellID \in OutCells$	$OldCellID \in NormalCells$
$NewCellID \in InCells$	情况一：workload 需要从 out 单元迁移到 in 单元	情况二：workload 需要从 normal 单元迁移到 in 单元
$NewCellID \in NormalCells$	情况三：workload 需要从 out 单元迁移到 normal 单元	情况四：workload 保持在 normal 单元中

Router 映射

- 对于情况四，如果两次映射都在 normal 单元中，则他们的一致性 hash 结果一定相等，此时将请求转发到对应的单元即可
- 对于情况一二三，代表 workload 需要被迁移，router 需要根据这次迁移操作是否完成来决定把请求转发到 OldCellID 还是 NewCellID, 此时的处理逻辑也很简单，只需要从 OldCellID 分片获取 workload 的 metadata:
    1. 如果 workload 的 rebalanceStatus lable是空，说明 Contoller 还没有开始当前对象的迁移，router 把请求转发给当前集群即可
    2. 如果 workload 的 rebalanceStatus label 值为 migrating，说明 Controller 正在对当前对象进行迁移，需要禁止对 workload 进行写操作，此时读请求正常转发，写请求返回重试即可
    3. 如果 workload 已经不存在或者 rebalanceStatus label 的值为 migrated，说明 Controller 已经完成当前对象的迁移操作，或者这个对象本身就不存在，此时 router 把请求转发到 NewCellID 所在集群即可

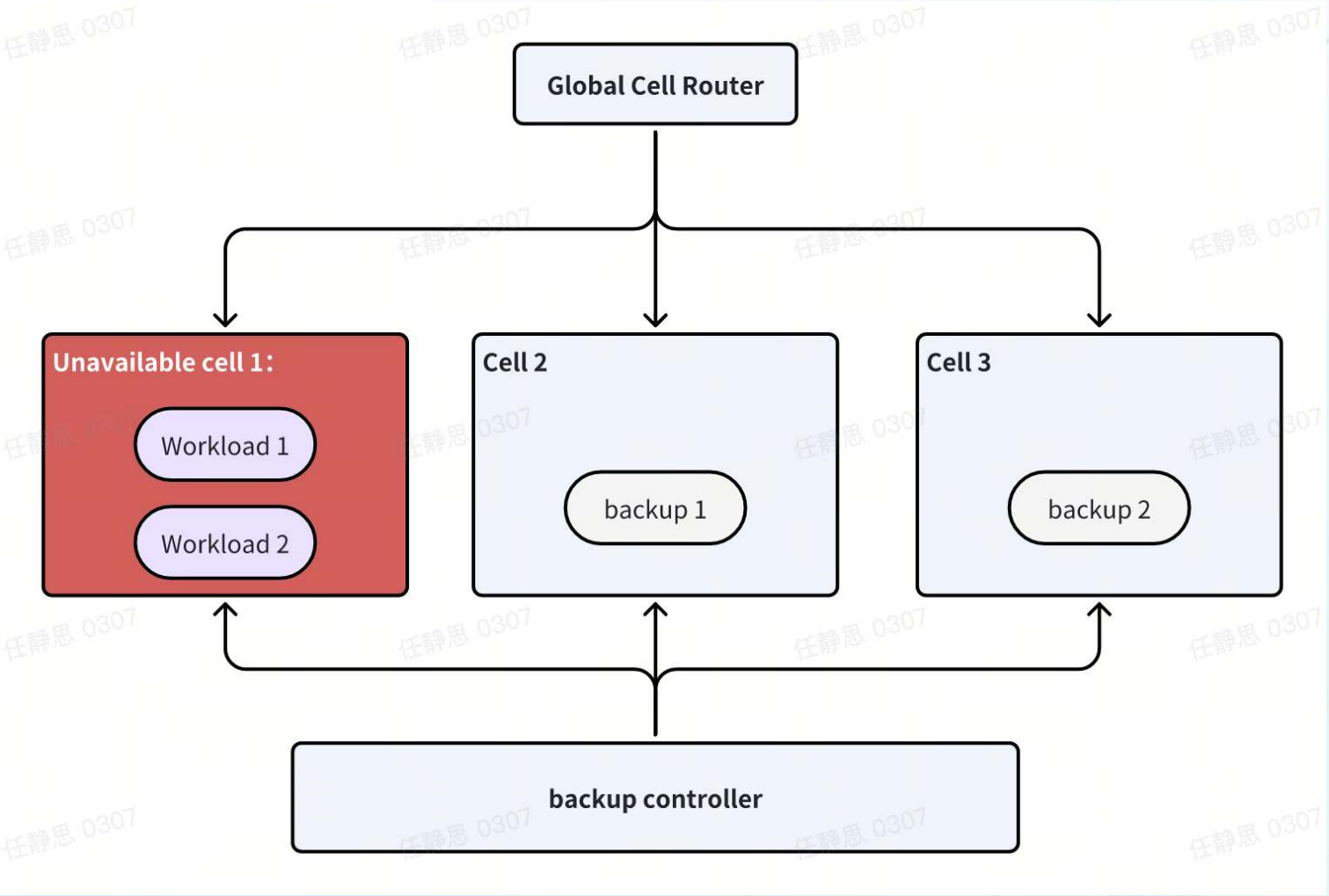
增加/裁撤单元

	$OldCellID \in OutCells$	$OldCellID \in NormalCells$
$NewCellID \in InCells$	情况一：workload 需要从 out 单元迁移到 in 单元	情况二：workload 需要从 normal 单元迁移到 in 单元
$NewCellID \in NormalCells$	情况三：workload 需要从 out 单元迁移到 normal 单元	情况四：workload 保持在 normal 单元中

Rebalance Controller

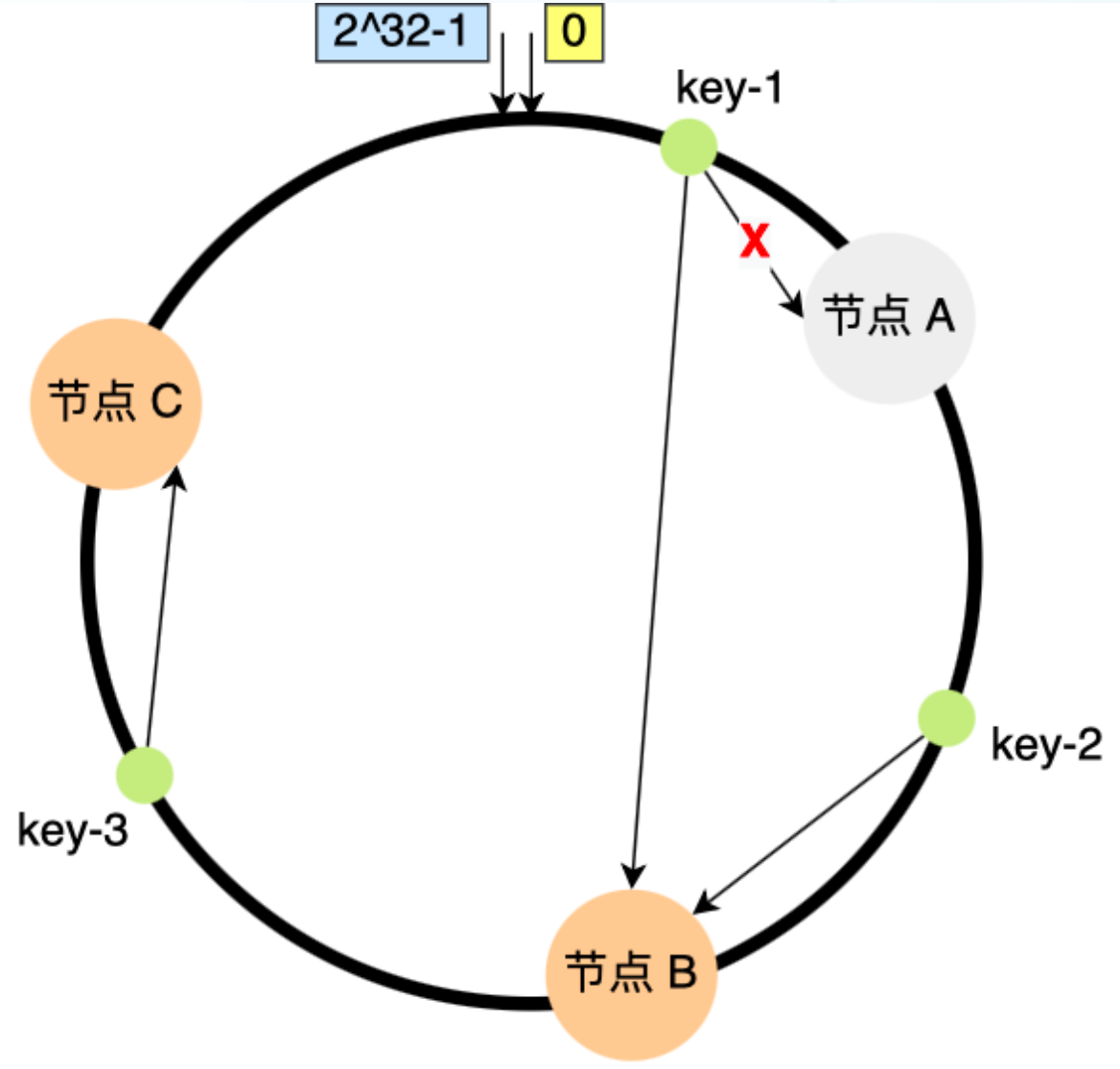
- 对于情况四，workload 无需 rebalance
- 对于情况一二三，rebalance controller 根据配置的迁移窗口大小，选取一定数量的 workload，进入如下迁移流程：
    1. 设置 workload 的 rebalanceStatus label 值为 migrating，此时可以保证 router 和 fed manager 都不会再更新该对象
    2. 把 workload 从 OldCellID 的单元复制到 NewCellID 的单元
    3. 更新 OldCellID 中的 workload rebalanceStatus label 值为 migrated
    4. 等待一定时间后（e.g. 12h），删除 OldCellID 中的 workload，完成迁移

单元间备份和容灾快速恢复



- 常态下资源对象只在对应的单元(主单元)内完成 federation controller 控制循环，我们引入旁路备份机制，通过备份控制器 (Backup controller) 实时把资源对象同步到一个备份单元中，在主单元发生灾难的时刻，可以更改配置把资源映射切换到备份单元中，快速恢复，保证服务不中断。

单元间备份和容灾快速恢复



- 一致性哈希恰好能应对这种单元数量减少的场景，在单元映射算法中，取一致性哈希的结果为 TargetCellID，取顺时针的下一个单元为 BackupCellID
- Backup controller 负责实时把主单元中的资源对象复制到备份单元中，并设置 backupCellID label
- 如果某个单元不可用
  - router: 在配置中去掉不可用的 cell, 请求会自动路由到 backupCell 中
  - backup controller: 根据最新的 cell 配置计算 cellID, 如果 cellID 和 backupCellID 相等，则删除 backupCellID label，把备份转正

Thanks.