**资源调度架构**

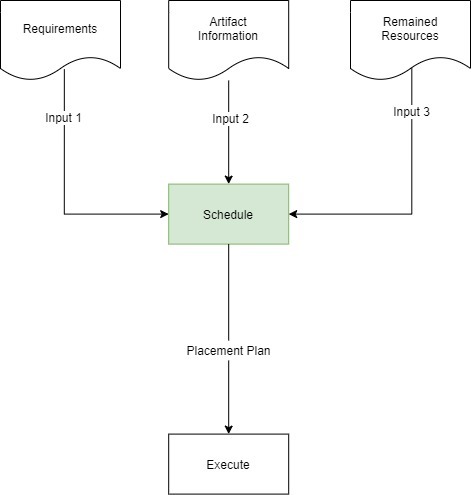
资源的管理策略随着参数向量的变化而变化, 参数向量的变化包括:

* 资源的物理变更
* 运维需求与策略的变化
* yae 自身架构的演进

在设计之初, 有意弱化了调度器的可控制性. 调度的条件只提供了 replica factor (副本数), need multiple idc (多机房分布) 两个. 一个原因是没有和资源的物理特征以及运维的需求产生联系, 另一方面也是为了确保调度器实现起来尽可能简单.

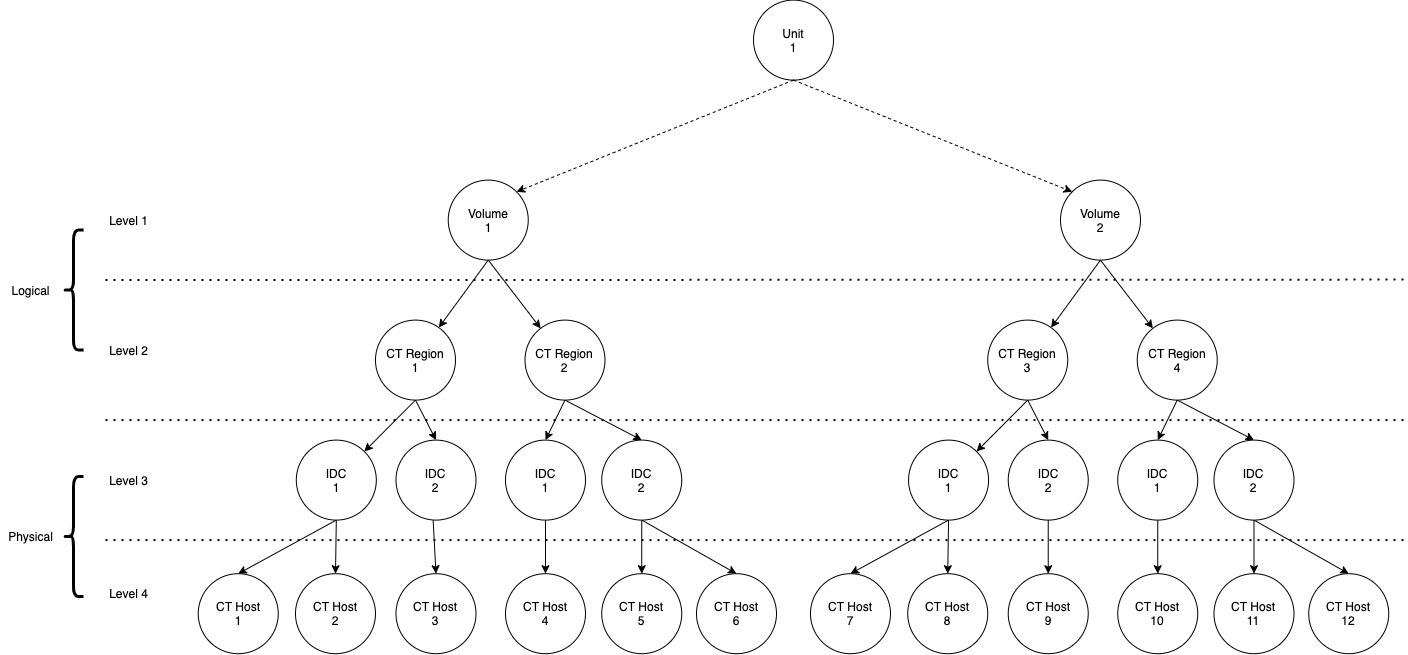
当调度条件开始动态多变, 目前的调度器显然已经不再合适了. 最大原因是灵活性收到制约. 因此, 在保证现有的调度正确性的前提下, 这里基于灵活性和可控性, 提出新的调度器模型.

不过有几点要注明, 新的架构并非对整个调度器做了完全的摧毁重建, 而仅仅是针对不合理的构件 (Schedule) 做了重新设计. 所以宏观的调度架构仍然保持原样:



我们首先对底层物理资源进行重新建模.

先将所有主机 (Host) 进行降维, 拍平成一维的线性结构. 随后将其本身固有的物理性质转化为他们的属性 (Attrs). 然后根据属性, 逐层向上以树型结构, 建立对物理资源管理的逻辑视图:



新的设计里, 正式引入此前被暂缓搁置的 SchedGroup (调度组), 现在称作 Volume (卷组), 提供更加可控的调度能力.

当加入到yae的物理资源以这种结构化呈现时, 调度器就可以遵循着特定规则工作了.

确切来讲, 资源管理系统形成的元数据是一个森林结构. 每棵树以一个确定的 Unit (单元) 为根.

在结构确定的前提下, 为了突出重点, 这里我们有意忽略运维层面设置的逻辑管理概念 Unit, Env, 只专注于以 Volume 为根的虚拟资源视图.

有了资源视图, 接下来就可以考虑调度了. 本篇所讲资源调度, 即在一个大小有限的虚拟资源树里, 使用贪心策略从当下资源状态快照里匹配恰好满足输入条件的叶节点 (Host).

当忽略最顶层两个组件, 我们可以把这个虚拟视图划分为四层, 每一层可以看作调度的一个阶段 (Phase).

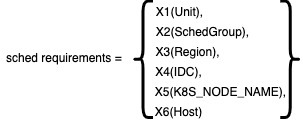
前两层是我们从资源管理角度出发设置的逻辑容器边界, 可以按照我们的意愿随意调整, 因此属于逻辑层.

向下两层是按照资源物理性质划分的边界, 不可变更和控制 (Immutable), 属于物理层. (其实 Unit 和 Env 从 yae 的视角, 也可以看作物理层, 因为它们不受我们控制且不可变).

需要注意的是, 树的第三层是 IDC, 其实 idc 只是最底层 host 的一个物理标签 (属性) 而已, 第三层本质上在做 "属性" 或者 "标签" 的匹配, 准确来讲这一层可以继续分为 3.1, 3.2, ... 但考虑到对标签的筛选目前只有 IDC, 所以图里用 IDC 这一个标签来代表这一层的调度对象.

当所有调度过程都遵循这个框架, 我们就可以明确定义调度需求 (Requirements) 了.

每个调度阶段都有对应的调度控制参数, 调度需求可以表达为调度控制参数向量.



直接配置这个Requirements Vector (RV) 并不是很搞笑, 所以我们会衍生出有实际意义且经常使用的 "副本因数" (ReplicaFactor (RF)) 以及 "多 IDC 分布" (Multiple IDC (MIDC)) 做为补充, 让调度器帮助我们去合理控制 RV 的值. RF 和 MIDC 分别用于控制调度的每一层如何下推生成 RV.X5 和 RV.X6 的值.

除了 RF 和 MIDC, 本次提案新引入一个 RV 辅助控制项: "非同源机器优先" (Non-Homologous Machine First (NHMF)). 这是一项基于 k8s 节点而衍生出来的调度控制方案, 因为我们拿到的 host 并不代表他们一定位于不同的 k8s 节点上, 如果配置了 NHMF, 就表示如果期望副本数超过 2, 那么调度器会尽可能让所有副本分布到不同 k8s 节点上, 进一步提高业务服务的灾备抗性. 这一过程类似 IDC, 也是借助 host 的物理标签 K8S\_NODE\_NAME 实现.

可以这么理解: 不论 RF 还是 MIDC 亦或是 NHMF, 都是为了方便用户去操纵 RV 而提供的 "高级配置" 方案. yae scheduler 充当一个函数, 将 RF, MIDC, NHMF 输入转换为一个 RV. 只有当我们需要对资源进行更细粒度的分配时, 才会直接使用 RV. 一般 RF+MIDC+NHMF 就足够了.

对于 RV, 还有另一种解构方式, 我们可以将调度输入条件分类为:

* 逻辑属性筛选条件. 如 Unit, Region, SchedGroup 等
* 物理属性筛选条件. 如 IDC, K8S\_NODE\_NAME, ENV, HostAlias 等
* 物理状态筛选条件. 如目标 Host 上现存 Node 数量, 过去 1h Load 等

RV 就是这样一组输入条件集合.

调度的最后一层, 对主机的选择. 经过了前几层的硬限制 (Hard Limit) 筛选, scheduler 对留下的 Host 完成最后一步调度.

这一过程对应主机物理状态的筛选. yae 平台可以定义一系列资源的阈值, 物理状态筛选时会参考这个阈值, 把相对过载的主机筛掉.

等所有的筛选条件走完, 调度器会拿到一个可用的 Host 集合.

如果前面通过 "高级配置" 生成 RV, 那么这一步将是整个调度器的核心, 也是以来 RF MIDC NHMF 工作的过程, 即对可用资源集合过滤 & 排序, 得到最能匹配调度条件的一组资源 (这个过程不在我们本次讨论范围里, 因此这里不展开, 后续单独开 RFC 讨论). 反之, 如果直接通过 RV 指定了 host, 则不存在这个过程了.

由此可见, 最终生成的副本放置方案 (Placement Plan) 即 RV 的子集.

执行器的角色就是拿着调度器给的放置方案部署副本, 没有特别的变化, 这里不再赘述.

最后, 我们这里所提到的调度器只是 Host 级别的调度, 而随着 yae vm 架构的升级, Node 级别的调度也会逐渐出现, 因此, 此番调整并非终点, 未来可预见会有两类资源调度模型共存, 而它们会遵循同一个宏观的调度框架.