应用混部研究方案

# 研究内容

中移在线基于自身需求，于2017年开始将应用容器化，并计划2018年对更多业务容器化。同时为了应对大规模生产环境中容器的管理和编排引入了Kubernetes。随着网络、存储、监控、镜像仓库统一管理平台的建设，如何充分利用容器技术来提升集群资源利用率的问题日益突出。对于万台规模的集群来说，资源利用率每提升1%，将为数据中心节省上千万元的设备费用。

针对中移在线应用的监控数据分析，结合应用在容器化改造时的经验特点，发现资源利用率方面存在下面痛点：

1. 应用资源使用呈现明显的高峰低谷；
2. 中移在线拥有大量的服务器资源，但资源利用并不充分；
3. 不同的机器的资源使用率差距较大。
4. 应用资源申请量和使用量之间差距巨大；

基于Kubernetes对集群资源池化的基础上，调度系统将不同类型的应用自动混合部署到整个集群，不受单个服务器边界限制。用这种方式来提升了资源利用率；同时根据集群内应用的负载情况自动伸缩应用实例，实现按量分配资源；当某些机器负载过高或者过低时，调度系统将迁移应用，均衡集群资源利用率。

# 技术方案

结合中移在线的痛点，设计出一套可以与Kubernetes系统配合的应用混部方案，抽象出下面四个调度场景：

1. 在线离线分时复用

**定义：**将资源利用曲线呈互补关系的在线和离线应用混合部署。

**效果：**应用混部通过实时数据和预测数据来判断在线的预计需要的资源使用量，当在线业务负载小，需要资源小，调度算法减小在线业务pod数，同时离线增加业务pod数，使得资源利用率一直处于高水位。

**验证：**在测试环境中让一个在线业务呈明细的曲线，其资源利用率也相应的上下波动，应用混部能根据波动来实时伸缩在线应用和离线应用，调整他们各自的资源配额。

1. 多在线应用混合部署

**定义：**将2-3个优先级不同的应用混合部署到相同主机上。

**效果：**应用混部通过历史监控数据，自动筛选出适合部署到相同机器的应用。 减少应用需要的集群数。

**验证：**在测试环境部署多个占用资源利用少的应用。应用混部自动将多个占资源少的应用集中到某几台机器中。

1. 资源消耗均匀化

**定义：**减小不同机器间资源使用率差异，避免某些机器负载过高或过低；

**效果：**应用混部通过历史监控数据，分析各个机器的资源使用情况，从资源利用高的机器迁移应用到资源利用率低的机器中，减小每台机器的资源利用率处于集群的平均利用率的差值。

**验证：**在测试环境中，部署多个占用资源不同的应用。应用混部将对应用进行迁移，均横各机器的资源利用率。

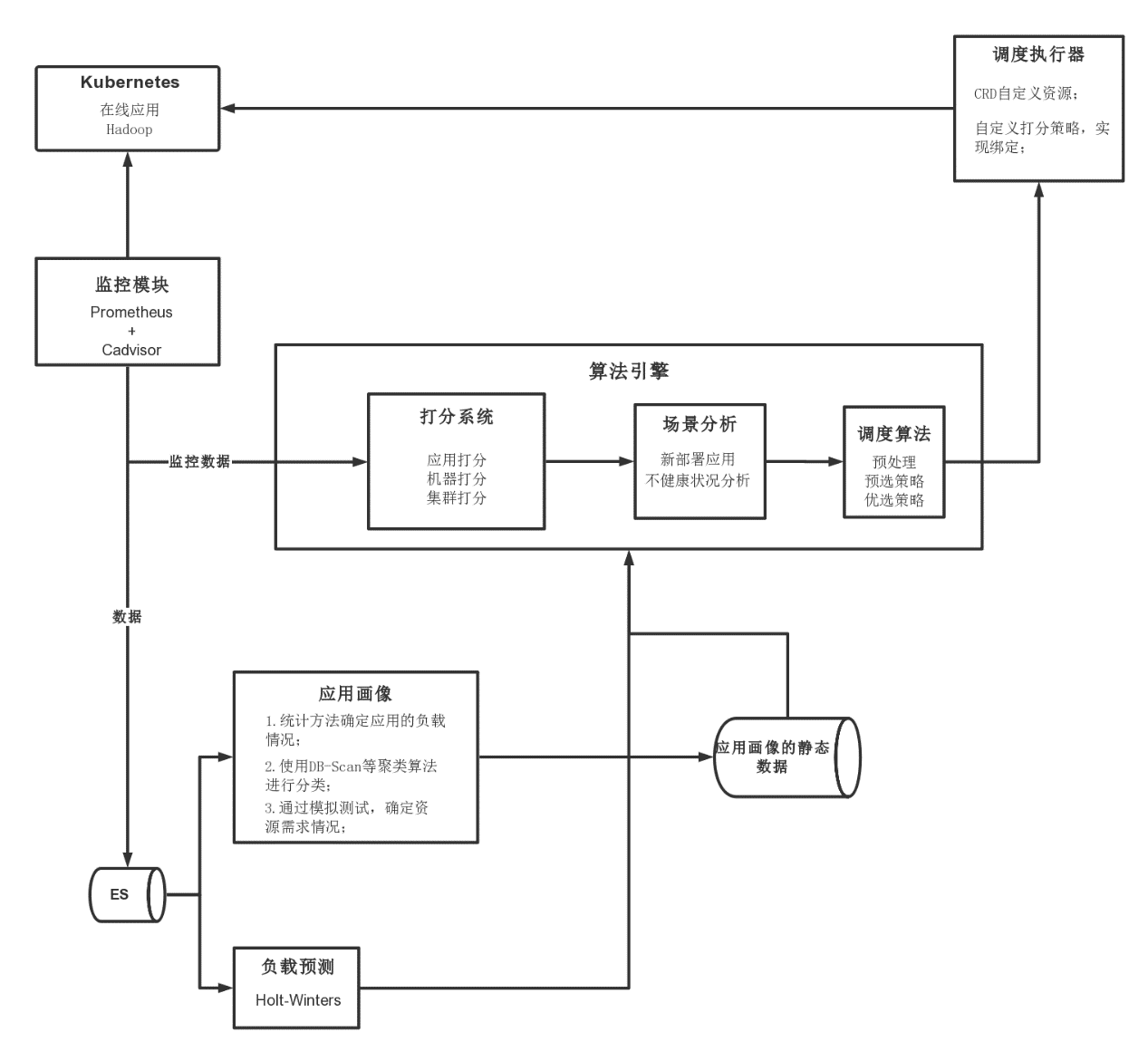
1. 调节应用资源配额

**定义：**调整应用的资源保留值(request)和最大值(limit)。

**效果：**应用混部通过历史监控数据，分析应用的资源使用情况，来确定应用合适的资源配额(保留值和最大值)。

**验证：**在测试环境中，部署实际资源使用和预定的资源配额有不同差异的的应用。应用混部将调整应用的资源配额。使应用的资源配额紧凑。

## 系统框架

****

上图展示了混部系统的整体设计框架，首先是监控系统Promethus，会实时收集应用、机器以及集群的信息数据，这些数据存储到Promethus的时序数据库中。

打分系统会根据监控数据以及预测数据对应用和机器进行打分。对应用打分就是判断该应用的性能指标(例如响应时间、错误率)是否达标，对机器打分就是根据机器的综合负载情况对机器进行打分。当综合负载在特定值时，分数最高，综合负载越高或者越低，分数就越低。

打分系统对所有的应用和机器进行打分后，场景分析模块会分析分数太低的原因，根据原因的不同，归类到不同的调度场景。

调度算法根据场景分析模块确定的调度场景计算出调度策略。调度算法是应用混部系统的核心，度算法会经过预处理、预选、优选等步骤，综合考虑资源约束、亲和关系、利用率均衡、分摊容灾、负载预测等因素，给出更好的节点选择。最终寻找一种最优的调度策略，给调度执行器执行。

所有策略生成完后，由调度执行器统一调用Kubernetes的API执行。

此外，为了能使混部效果好，混部系统引入了应用资源画像和资源预测。由于这两个模块都需要大量的历史数据，所有定期的将有用的监控数据存储到数据库中，应用资源画像和资源预测这两个模块直接去数据库中获取相应的数据。

所谓应用资源画像就是根据应用的历史数据，构建出该应用的大致资源使用，负载情况等的轮廓。这些数据也会存储到数据库中。

负载预测模块主要是根据在线服务的历史负载数据，对该在线服务接下来一段时间的负载情况进行预测。因为调度有延迟，如果等在线服务的服务质量下降了再去调度，可能会由于调度延迟带来很多损失，因此引入了负载预测，进行提前调度，规避服务质量的下降。由于一般在线服务的负载都是具有明显周期性的，将选用Holt-Winters算法对负载进行预测。

## 模块介绍

下文将详细介绍各个模块功能和实现方式。

### 监控模块

应用混部的基础是大量的监控数据，只有监控数据完备，才能分析出集群的资源利用状态。Kubernetes上最合适的监控系统是Prometheus，也是为数不多的适合Docker、Mesos、Kubernetes环境的监控系统之一。

Prometheus基本原理是通过HTTP协议周期性抓取被监控组件的状态，这样做的好处是任意组件只要提供HTTP接口就可以接入监控系统，不需要任何SDK或者其他的集成过程。这样做非常适合虚拟化环境比如VM或者Docker 。

与其他监控系统相比，Prometheus的主要特点是：

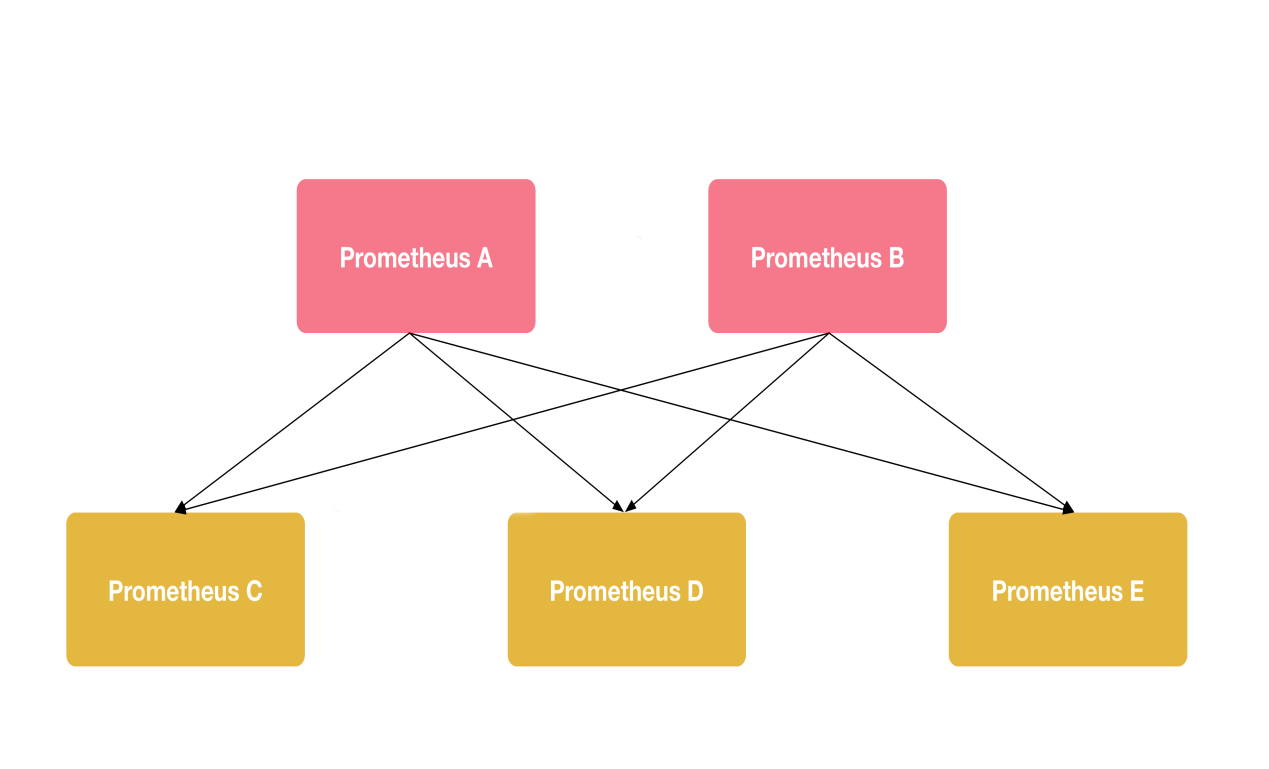
1. 一个多维数据模型（时间序列由指标名称定义和设置键/值尺寸）；
2. 非常高效的存储，平均一个采样数据占3.5bytes左右，320万的时间序列，每30秒采样，保持60天，消耗磁盘大概228G；
3. 一种灵活的查询语言；
4. 不依赖分布式存储，单个服务器节点；
5. 监控数据通过HTTP上的Pull模型获取；
6. 通过服务发现或静态配置发现目标。

#### 监控高可用

因为Prometheus的监控是单节点模式，在应对实际生产环境时，必须做到高可用，我们用过下面两种方式实现高可用：

**基本HA + 集群联邦**

首先通过底层Prometheus节点(黄色)对机器或指标进行分片收集，最后汇总到上层Prometheus节点(红色)，为了高可用上层节点可以配置多实例。



**远程存储**

虽然Prometheus有自己的数据存储机制，但因为Prometheus的特性，其对在内存的热数据查询非常迅速，但是对于已经落盘到磁盘的老数据查询效率并不突出。为了供应用画像，负载预测等模块查询数据。我们可使用独立的数据库来存储历史数据。

#### 监控维度

在 Prometheus 中可用静态的方式来监控 Kubernetes 集群中的普通应用，但是如果针对集群中众多的资源对象都采用静态的方式来进行配置的话显然是不现实的。针对应用混部所需的数据监控程度，将监控级别划分为四类，分别使用相应的Exporter对其进行监控，具体见下表所示：

|  |  |
| --- | --- |
| **维度** | **Exporter** |
| 主机 | node-exporter |
| 服务 | Ingress |
| 容器 | cAdvisor |
| 集群 | kube-state-metrics |

在各监控级别上，针对不同的资源类型，可以将资源监控类型划分为四类，利用上述工具监控相关的指标，具体指标类型见下表所示。

|  |  |
| --- | --- |
| **资源监控类别** | **监控指标** |
| CPU | CPU利用率最大、最小、平均值 |
| 内存 | 内存利用率最大、最小、平均值 |
| 存储 | 磁盘/存储卷使用量、IO速率 |
| 网络 | 占用带宽，吞吐量，连接数，延迟时间 |

### 负载预测

混部调度系统往往是根据实时的监控数据来判断应用以及机器的状况，从而来决定是否执行调度。但由于调度需要一定的时间，往往启动或者迁移一个pod需要一定的时间，存在一定的调度延迟，这将导致在调度生效的这一段时间内，可能发生服务质量的下降或者机器负载偏高等情况，造成不必要的损失。此外，实际环境中经常会出现服务质量或者应用负载发生抖动，如果只凭当前的监控数据去判断，容易造成误判造成一些不必要的调度。

对负载进行预测，往往能避免上面提到的情况。因此系统中引入了负载预测模块，对负载进行提前预测，调度系统同时对实时监控数据和预测数据进行分析，得到更精准的调度策略，保证提前调度和去除应用抖动等。

#### 预测方法

常用的负载预测方法有很多，有基于机器学习的负载预测方法，这类方法当数据比较充足时，一般比传统的方法要准确，但效率一般很低，实时性很低，一般实时分析的系统不采用这种方法。还有就是基于时间序列的负载预测方法，这类方法将负载看做成一个时间序列，利用ARIMA，Holt-Winters等时间序列预测方法进行预测，这类方法对具有周期性，趋势的数据预测效果一般很好，但是参数很难确定。

结合实际场景，分析出中移在线大量应用的负载都具有明显的周期性，采用基于时间序列的负载预测方法更合适。经过简单的测试实验，发现ARIMA在参数准确时往往具有较好的效果，但是参数很难确定，往往要经过大量的人工数据分析，才能确定较为恰当的参数。所以这个方法扩展性不高，一般不使用，Holt-Winters是一种专门做周期性的时序数据的预测方法，该方法参数扩展性很高，效果也很好，所以最终选用这种方法作为该设计的负载预测方法。

#### 模块设计

预测负载模块是根据历史数据利用负载预测方法对应用的负载进行预测。为调度算法分析提供相应的数据支撑。本模块牵扯到大量的计算，所以设计为单独的模块。在大规模混部系统下，牵扯到大量的计算，单台机器是不行的，所以该模块设计为集群的方式，通过spark集群为该模块提高计算资源，保证该模块不是系统的性能瓶颈。

对于每一个应用，本模块周期性的启动一个spark任务，从存储历史数据的数据库中获取相应数据的过去n（>10）个周期的数据，用这些数据作为预测算法的输入，预测出该应用的下一个时间段的负载情况，并将预测数据存入数据库中，调度算法模块需要时直接去数据库中查询预测结果即可。

### 应用画像

应用画像主要承接了监控平台采集的数据源，通过对各应用一定周期内的历史数据做出分析，构建各应用和服务的细粒度资源需求画像，为打分模块和调度算法提供支持。

#### 模块设计

应用画像需要的数据来源主要由监控平台Prometheus监控采集，主要通过下面四类数据：

1. cpu资源：container/pod/service/node的cpu利用率；
2. 内存资源：container/pod/service/node的内存利用率；
3. 磁盘资源：container/pod/service/node的磁盘I/O速率；
4. 网络资源：container/pod/service/node占用网络带宽；

结合中移在线应用容器化到Kubernetes时的部署特点，可以通过应用-服务-实例拓扑关系，来表示应用等级关系。

应用画像通过将Prometheus监控采集到的各应用和服务在一定周期内的历史数据做初步处理后，使用K-Means、DBSCAN、EM等聚类算法对各应用所属类型进行聚类，进而区分出CPU密集型、内存密集型、磁盘密集型、网络密集型等资源占用类型。同时会将应用服务、应用实例数、资源配额等静态信息作为应用的特征。

通过画像算法和统计方法对应用的历史数据分析，可得到该应用下服务的负载分布区间、最大负载、最小负载、平均负载、变化趋势、及相应资源维度下的分布。

通过对应用每个服务整体的资源维度画像，衡量pod中的资源维度间的关系。然后利用SearchOnInit等相关算法计算能使对应服务达到相对紧凑状态的pod各维度分配值，并通过模拟不同负载情况得到需要需要提供相应pod支撑服务质量的数量。

#### 应用特征

经过应用画像后，就可以通过下面这些特征来描述每个应用，这样每个应用都能被统一标准的描述了：

1. 应用的层级结构，应用由哪些服务组成，那些pod实例组成服务等；
2. 应用的优先级；
3. 服务所属类型(CPU密集型，内存密集型，磁盘密集型，网络密集型等)；
4. 服务在周期内各时间段的负载变化趋势、平均负载、最大最小负载；
5. 服务在周期内各时间段不同负载情形下资源占用情况需要的相应pod实例数；
6. pod在启动时，无负载和不同负载情况下资源占用情况；
7. pod镜像拉取时间、pod的启动时间；
8. pod的资源配额：CPU和内存的request，limit值；
9. pod 的亲和性、反亲和性、标签值等；
10. pod的存储卷类型、大小。

### 打分系统

打分系统主要功能是通过来自Prometheus的应用和机器的监控数据和应用负载预测数据，对应用或者机器打分，判断其是否需要调度，为场景分析模块的处理依据。

#### 打分规则

打分系统首先通过下面三个维度对这个集群打分：

1. 对于应用的打分:对于应用的打分，我们是根据应用的一些性能数据(例如：请求的速率、错误、耗时等)对其进行综合打分，例如某个应用的性能数据超出了我们设定的阈值，则直接将其分数打的非常低，如果性能指标正常，我们再会根据应用的负载以及所占资源是否匹配来进行打分，例如如果分配给该应用的资源远远超出所需资源，则将其分数打到很低；
2. 对机器的打分:跟据机器的资源情况为该机器打分，；
3. 对集群的打分：主要是达到使整个集群资源平均的目的，打分方式是求出每个机器使用资源和总资源的比值，然后对所有机器的这个值求方差，判断他们的离散程度，方差越大，集群打分越高。

除了上述三个基本打分方式外，还可以根据集群实际情况来修改默认规则的阈值、权重、占比等。

### 场景分析

场景分析模块主要提供两个模块，一个是支持新增应用，当要在集群中部署新的应用时，调用该模块来确定该应用所需的实例数以及每个实例的部署节点；另外一个部分是根据打分系统的打分数据，判断应用，机器以及集群的分数是否在正常范围之内，如果不在，则进行分析不健康的原因，确定调度场景(新增实例，删除实例，迁移实例等场景)。

#### 新部署应用

当集群上部署新应用时，首先在数据库中获取到该应用的应用画像数据，包括该应用下面的服务，每个服务的负载情况，以及每个服务的空闲时资源使用情况，启动时资源使用情况等等。

对于该应用的每一个服务，根据应用画像数据确定它在大部分时间下的负载情况，以及这些负载所需的资源情况。然后根据分配的资源情况开确定pod实例个数。

#### 分析集群分数

当打分系统完成打分后，则会调用该模块来判断应用，机器以及集群的分数是否在正常范围之内，一般是我们根据实际场景设定固定的阈值，当超过阈值分为，我们判定为需要调度。

对于分数低的应用，需要知道是应用负载过高或应用负载过低，或者受其他应用的影响，亦或者是有些不知道的因素中的那种原因导致应用的分数过低。首先需要获取到该应用当前的负载数据，然后根据应用画像数据计算当前负载情况下应用的需求资源，在对需求资源和实际使用资源做对比：如果需求资源远远大于实际使用资源，则判断应用服务质量下降是因为为本身负载过高导致，则确定其为服务扩容场景；如果需要资源情况远远小于使用资源情况，则确定其为服务缩容场景；如果前两者都不是，则分析该应用下的所有pod情况，如果部分pod的服务质量正常，部分不正常，则分析那不正常的几个pod的本身所在机器的负载情况，如果是因为本身机器的负载过高，则确定为机器降负载场景；其他则确定为未知因素场景，报警。

对于每个分数低的机器，就是判断自身负载情况，如果太高，则为机器降负载场景；如果是负载太低，则确定为机器升负载场景；其他的则为机器未知因素场景，报警。

如果集群分数处于需要调度范围，则确定为负载均衡场景。

判断出不同的调度场景后，根据不同的场景，场景分析模块的处理方式分成下面几种情况：

1. 对于服务伸缩场景：首先会根据该服务的负载预测值判断负载变化是一个抖动还是会持续，对于抖动情况不做处理；对于可能持续的情况得到下一个时间段的负载值，再通过应用画像数据，确定伸缩的实例个数后，交给调度算法生成调度策略；
2. 对于机器升降场景：首先会根据该机器上全部pod的负载预测值判断机器负载变化是一个抖动还是会持续，对于抖动情况不做处理；对于可能持续的情况通过计算集群内机器下个时间的负载情况，确定那个pod会被迁移后，交给调度算法生成调度策略；
3. 对于机器均衡场景：交给调度算法生成调度策略；
4. 对于未知因素场景：直接发出告警，让人工干预。

根据实际的场景中根据应用不同，还会有很多情况属于的特殊调度场景。用户可以根据自己的实际业务来编写适合的场景分析模块，注册到场景分析模块来扩展分析功能。

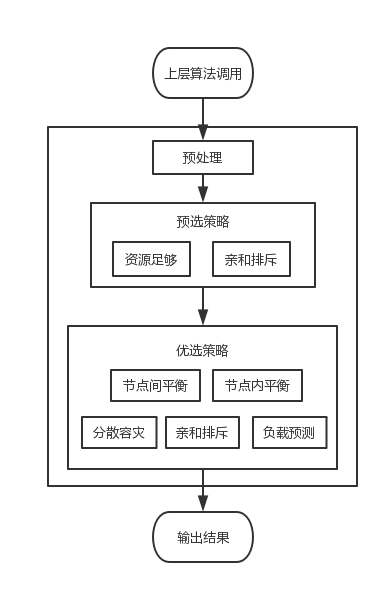
### 调度算法

调度算法主要工作是在场景分析出的结果上再加工。确定调查的调度顺序，用最小的步骤和资源完成需要调度的pod，调度算法主要处理下面几种调度情况：

1. 添加pod：确定哪些pod需要被添加到哪些机器上；
2. 删除pod：确定哪些机器上的哪些pod需要删除；
3. 迁移pod：确定哪些机器上的哪些pod需要迁移到哪些pod上。

#### 算法思路

算法要求用时短，时效性高，无法做耗时长的优化，所以采取类似于Kubernetes自带的策略，同时针对我们的场景进行一些定制和改进。



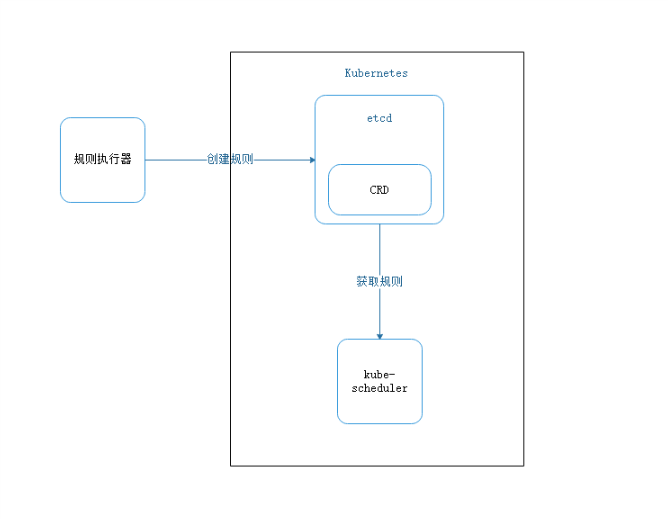
1. 预处理，对要增加的pod进行排序，按照占用资源从大到小的顺序。这一步参考了类似First-fit decreasing算法的思想，先用大pod平衡机器间的差距，再用小的pod微调，；
2. 按顺序为每一个pod评估各个机器。首先是预选，筛选出此pod可选的机器，筛选依据有：检查节点是否有足够资源，应用之间强亲和/反亲和关系、应用宿主机强亲和/反亲和等。这些筛选依据是必须满足的硬约束，不满足的机器直接排除；
3. 其次是优选，对预选阶段得到的可选机器逐个打分，打分策略会考虑：优先使用空闲的节点;尽量使CPU内存等利用率一致，避免出现类似CPU很高但内存很低这种情况；为了容灾考虑，尽量不把同一服务下的pod放在一起；根据pod间的亲和性和反亲和性；通过负载预测得出下一段时间内机器上各个pod的负载变化情况，计算出机器的未来的负载变化，算法也会考虑这个变化。

通过上述约束来生成最终的调度策略。通过自定义的Kubernetes API调度算法可以将调度策略存储到etcd中，供调度执行模块获取调度策略。

### 调度执行

为了保证调度的效果，需要将某个应用的pod准确的调度到某个机器上，而Kubernetes目前还不支持这个功能，为了实现精准调度，需要修改Kubernetes的调度器，而Kubernetes支持自定义，只需要将新的调度逻辑注册到Kubernetes就可以。

#### 模块设计



上图是模块的功能图。包含两个模块，一个是调度执行器，一个是kubernetes集群定制化设计。

**规则执行器**

这个模块的功能是执行调度策略。调度执行器从调度算法获取到调度策略，然后操作kubernetes集群，对集群做具体的调度。在整个过程中，调度执行器的输入是一条条的调度策略，输出是对集群做的操作，以及自定义的的策略资源。

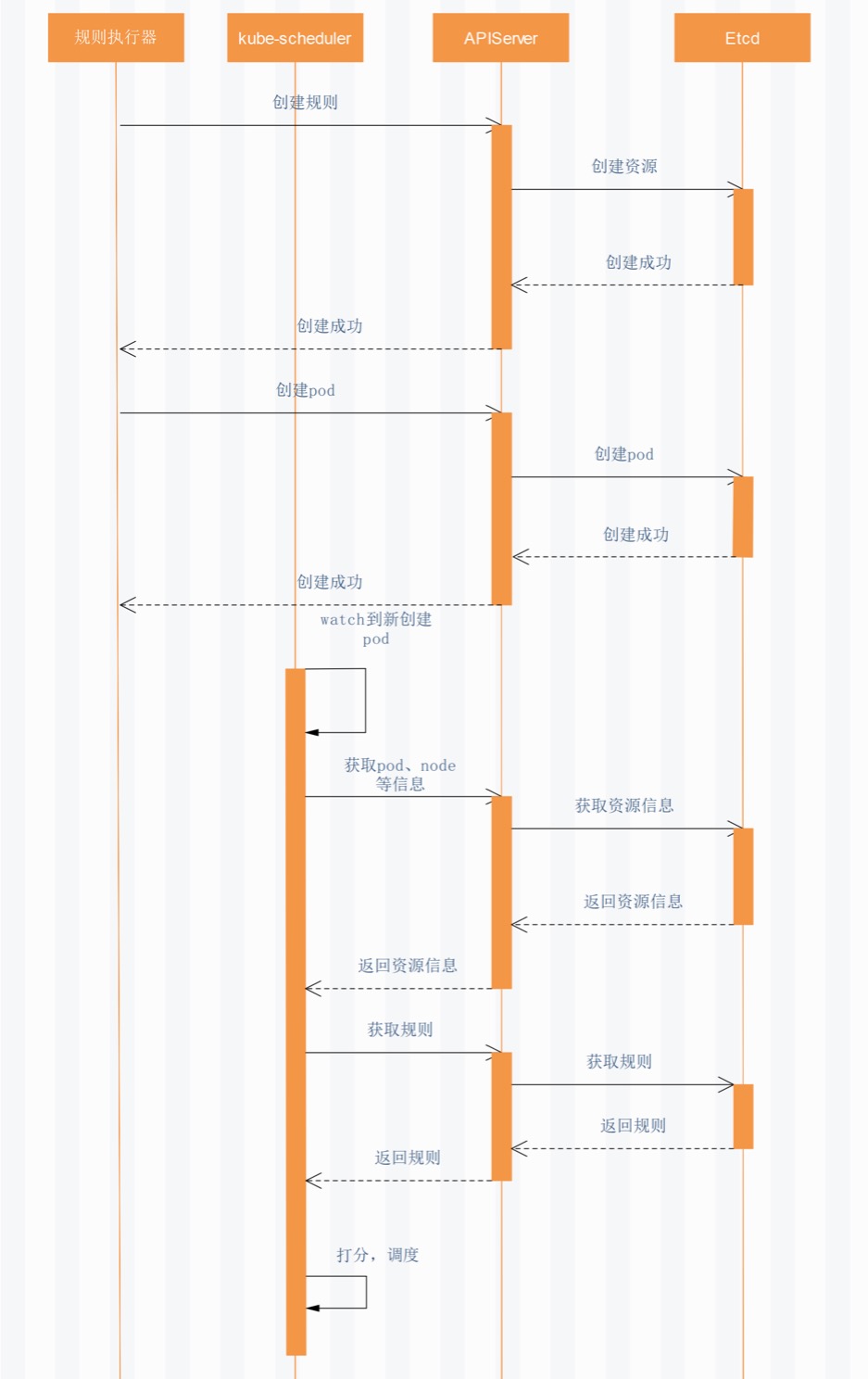
**Kubernetes策略**

这个模块包含两个部分，第一个部分是自定义的kubernetes的策略资源，第二部分是自定义kube-scheduler优选策略。

自定义kubernetes策略资源包含两个部分，一部分是调度策略的载体，即CRD，另一部分是CRD对应的控制器，实现对CRD资源的监控和维护。

自定义kube-scheduler优选策略主要实现的功能是将pod绑定到机器上。该功能主要通过给kubernetes默认的调度器添加一个优选策略的方式实现。将我们想要调度到的机器打最高分，从而实现将pod调度到特定机器上。

各组件之间的协作图——以新建一个pod为例



1. 调度执行器收到调度算法发来的调，度策略数据后首先调用apiserver的API创建一个CRD资源;
2. 该资源承载了调度策略的信息，然后apiserver将资源信息存到etcd中，返回创建成功信息。
3. 资源创建之后，调度执行器调用API创建一个pod，apiserver将pod信息存到etcd中，返回创建成功信息;
4. kube-scheduler的watch机制检测到etcd中有未分配机器的pod，从etcd中先后获取pod信息和CRD信息;
5. 在优选策略执行过程中检测pod信息和CRD信息是否匹配，匹配则将机器打最高分，否则不干涉默认调度器的执行。

# 下一步计划

在10月份，我们通过一个demo验证了k8s能支持应用混部。下面的计划：

**11月份**

设计混部应用方案，理清各个模块功能，设计实例；

测试各种应用画像，集群打分，场景分析，调度算法模块的最佳实现方式；

**12月份**

实现整体设计方案的整体框架实现，对仿真数据进行仿真调度；

**1-2月份**

优化算法，保证算法的稳定性，准确性；

**3-4月份**

最后在实验环境对真实应用进行调度，验证调度算法效果；