4.6 基于资源画像的调度策略实现

（总结前三小节的内容，上述内容为本章节服务）

本文通过“内存子系统压力定义”，“硬件性能指标”，“应用资源画像”三个部分的工作实现了对应用的实际资源需求的感知，包括 CPU，内存子系统竞争度。本小节将上述的工作结合起来，应用到调度策略中，使调度系统可感知应用实际资源负载，实现集群资源负载更加平衡，减少低负载节点与高负载节点的比例，从而达到减少资源浪费与应用服务质量下降风险的目的。

4.6.1 资源负载感知的策略实现

4.6.1.1 基于容量推荐的预选策略优化

（介绍方法）

预选策略的作用是根据一定的约束条件，预选筛选出能够满足 Pod 调度的一部分节点。这里介绍最值得关注的基于资源的预选策略 PodFitsResource。

PodFitsResource：根据 Pod 资源需求(Request)，与 NodeInfo 的可分配的资源(allocatableResource)比较，若 NodeInfo 的可分配资源无法满足 Pod 的资源需求，则该节点将被筛除。

预选策略中，从资源角度，本文将结合容量推荐优化 PodFitsResource 策略，实现更优的资源超卖策略。前文提到了 Pod 的资源需求(Request)是由业务方设置，资源需求与应用实例实际需求差异通常过大。通过容量推荐，调度器可对应用实例各时段的资源需求上限有所感知，所以将该资源画像信息引入到预选策略中，能够实现更加合理的资源超卖。

（本文作出的改进与创新：引入指标：资源画像的容量推荐与静态超卖率）

在 Kubernetes 原生的 PodFitsResource 策略做了优化，引入了应用资源画像的容量推荐与静态超卖率 OverSell。应用资源画像的容量推荐的引入是为了避免 Pod 请求资源设置不合理带来的资源浪费，作为资源超售的软限制。静态超卖率的引入是为了避免应用资源画像的部分不准确引起的节点严重超售，作为资源超售的硬限制。

本文优化的预选策略具体如图 错误!**文档中没有指定样式的文字。**.1 所示，本优选策略的关键思路分软限制与硬限制两方面:

（策略的关键限制条件）

软限制：假设 Pod 已调度到节点，即调用 AssumePodOnNode 后(该函数逻辑在小节**错误!未找到引用源。**中介绍)，获得 Pod 调度到节点后的画像数据。满足节点 24 小时画像数据中的 cpucap/memcap 都不超过节点的实际资源量。

硬限制：假设 Pod 已调度到节点，即调用 AssumePodOnNode 后，节点已分配的 CPU 资源不超过节点基于超卖率缩放的资源总量，以及节点已分配的内存资源不超过节点的实际内存总量。

（通过伪代码展示预选算法）

Algorithm 1 PodFitsResourceOnPortrait(P, N)

```
/*The function returns false if the node fails to satisfy the resource request from
pod, otherwise it returns true */
/* P is the pod */
/* N is the nodeinfo */
begin
  pnr ← AssumePodOnNode(P, N)
  cl ← pnr.resourceCapacity.CPU
  ml ← pnr.resourceCapacity.Memory
  lp ← pnr.resourceCapacity.loadsOnPortrait
  if pnr.requestedResource.CPU > cl * N.OverSellRate or
  pnr.requestedResource.Memory > ml then
    return false
  end if
  for h ← 0 to 23 do
    if lp[h].cpucap > cl or lp[h].memcap > ml then
      return false
    end if
  end for
  return true
  end
```

图 **错误!文档中没有指定样式的文字。**.1 PodFitsResourceOnPortrait 预选算法

4.6.1.2 基于高斯百分位估计的优选策略优化

（方法介绍）

Kubernetes 调度的原策略在“2.1.2 Kubernetes 调度逻辑”中有过描述，目前本文所在公司采用的优选策略为 LeastRequestedPriority，该调度算法的思路比较简单，即优选已分配资源较低的节点。而前文已有描述，Kubernetes 调度考虑的并非应用的实际资源需求，这样会带来的缺陷是集群资源负载分布不均衡，容易出现负载偏高或偏低的节点，偏高的负载会引起应用服务质量下降，而偏低的节点则影响集群的资源利用率。本文基于概率分布实现了新的优选策略，使集群节点间的负载更加均衡。

（本文的创新：通过加权平均实现整体优化）

本文对于三种资源(cpu, llcmiss, crpki)都采用相同的优化思路，不过由于数据单位不同，本文采用实现三个子策略与加权平均的方式实现整体的优化方案。

本文优化的核心思路是将“资源消耗特征”计算出的应用资源分布数据引入调度策略。随后基于正态分布的定义与其概率累积函数，再引入经验定义的阈值，实现集群中节点资源均衡化的目的。

（算法实现过程介绍）

算法的实现如图 **错误!文档中没有指定样式的文字。**.2 所示，由于三类资源的优化思路都是一致的，所以不分别赘述，以统一的方式描述算法实现。算法定义了负载预期值T，即对于当前资源r，期望节点的负载接近该负载。算法调用函数$AssumePodOnNode(P,N)$，获取应用实例P调度到节点N后的基于画像的资源分布情况pnr，其中基于画像的 24 小时负载数据为lp，遍历lp，取出基于画像的负载最高的关键数据记为$mean,std$，即分别为新资源高斯分布的平均值与标准差。根据本文对P调度到N上后，N的资源分布仍为高斯分布的假设，此时资源值r服从高斯分布$N(mean,std^2)$。算法目的是希望节点的负载快速达到与聚集到期望值T，所以直接依靠期望值T与$mean$两值之间的概率大小$P(T<\mathfrak{r}<mean)\ \ or\ \ P(mean<\mathfrak{r}<T)$作为节点分数的依据，具体思路是当$mean$小于$T$时，$0.5+P(T<\mathfrak{r}<mean)$作为该节点此次调度该资源维度的分数，即当前节点负载越小，优先级越高；当$mean$大于T时，$0.5-P(mean<\mathfrak{r}<T)$作为该节点的分数，即当前节点负载越高，优先级越低。由正态分布的特性可知，$mean$的左半边与右半边，概率都不会超过 0.5，所以对于节点低负载的情况时，采用$0.5+P(T<\mathfrak{r}<mean)$作为分数；而节点高负载的情况时，采用$0.5-P(mean<\mathfrak{r}<T)$作为分数；目的是保证低于$T$负载的节点的所得分数一定高于$T$负载的节点，由此快速填充低负载节点，实现集群跨节点资源更加均衡化，算法示意如图 **错误!文档中没有指定样式的文字。**.3 所示。

所有参与调度的节点与当前调度的 pod，对三类资源都采用上述算法进行打分后，各节点n在各资源r都有相应的得分$score[r][n]$，对各资源的分数$score[r]$进行归一化计算$Norm$(本文采用 Min-Max 归一化方法)，最后计算出各节点n的最终分数如公式(4.7)所示。调度器将选择最高分的节点绑定 pod，其中ω为资源的权重，本文将三类资源权重都设置为 1，另外使用 Kubernetes 的 ConfigMap 支持权重ω与预期值T的热更新。

$$finalScore[n]=\sum_{r\in\{cpu,llcmiss,crpki\}}\omega_r*Norm(score[\mathrm{r}][n]) \tag{4.7}$$

（通过伪代码展示预选算法）

Algorithm 2 ResourceBalanceOnPortrait(P, N)

/* The function returns score of node */
/* P is the pod */
/* N is the nodeinfo */
/* r is the resource name, including cpu/llcmiss/crpki */
/* T is the target load, defined in empirical */
/* $ProbabilityBetween(\mu, \sigma, v_1, v_2)$ returns the probability between value v_1 and v_2 based on the normal distribution of $N(\mu, \sigma^2)$ */

begin
  $mean, std, score \leftarrow 0$
  $pnr \leftarrow AssumePodOnNode(P, N)$
  $lp \leftarrow pnr.resourceCapacity.loadsOnPortrait$
  for $h \leftarrow 0$ to 23 **do**
    $m \leftarrow lp[h].r.mean$
    $s \leftarrow lp[h].r.std$
    if $m > mean$ **then**
      $mean \leftarrow m$
      $std \leftarrow s$
    end if
  end for
  if $mean > T$ **then**
    $score \leftarrow 0.5 - ProbabilityBetween(mean, std, T, mean)$
  else
    $score \leftarrow 0.5 + ProbabilityBetween(mean, std, mean, T)$
  end if
  return $score$
  end

图 **错误!文档中没有指定样式的文字。**.2 ResourceBalanceOnPortrait 优选算法

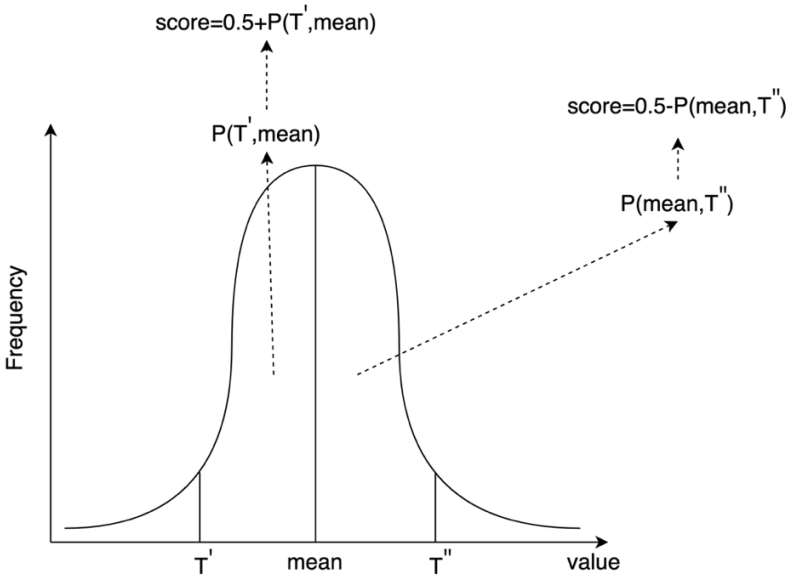

图 **错误!文档中没有指定样式的文字。**.3　高斯估计算法示意图