统一资源调度框架综述

# 摘要

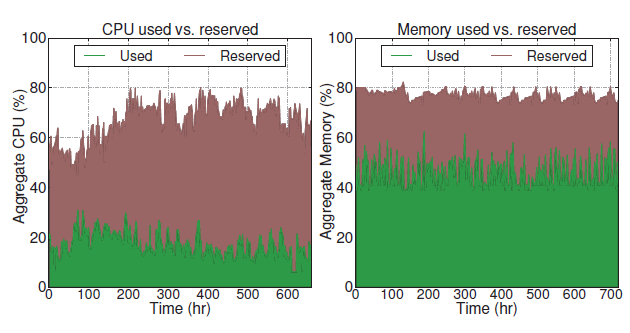
为节省企业成本，越来越多企业开始做在离线混部的工作，本文主要是对统一资源调度框架相关背景知识与技术做介绍。首先，本文介绍了在离线混部的背景，即企业在数据中心的投入越来越大，但利用率一直较低。随后，本文介绍了两个业界在离线混部的实践，分别是Google Borg，腾讯Caelus，两个系统的不同是前者是完全统一资源管理，而后者引入了协调层管理资源。再随后，本文介绍了混部会涉及的关键技术，如资源预测，应用亲和性等技术构成应用画像模块；调度算法与调度器架构构成调度模块；数据收集，干扰检测与处理等技术构成了节点Agent模块。最后本文总结了在离线混部的主要挑战，内存带宽隔离与统一资源管理框架。

关键词：在离线混部、统一资源管理、应用画像、内存带宽隔离、干扰检测与处理

# 一、背景介绍

随着云计算的兴起，越来越多的公司将他们的服务迁移到了云上，在云计算的范式下，用户只需要提交自己的服务，而不需要去关心服务到底在哪里运行。云计算依赖的是一种被称为仓储规模计算机的大型数据中心，这些数据中心由几万台到十几万台服务器组成，与云计算兴起形成反比的是规模不断增长的数据中心的资源利用率偏低[1]，仅为10%~20%，较低的资源利用率使得数据中心的成本收益率较低。据估算拥有100万台服务器的数据中心建设成本在260亿元左右，其中购买服务器的成本在100亿元左右，同时数据中心还有服务器维护和更新成本，每年的运维成本在40亿元左右，假设这100万台服务器的资源利用率为20%，那么每年的运维成本中32亿都被浪费掉了，假设这时数据中心的资源利用率提高了50%，那么原来100万台服务器才能完成的工作现在只需要67万台服务器，节省的购买服务器成本和运维成本为39亿元。百度是国内最早注意到数据中心利用率问题的公司，他们的Matrix集群操作系统能有效提高集群资源利用率，据百度估计仅2014年一年，Matrix的使用就为百度节省了6亿元。由此可见提升数据中心资源利用率对降低企业数据中心持有成本TCO的重大意义。

为什么数据中心的资源利用率一直偏低？这是因为大部分数据中心内运行的都是在线作业，这是一种对访问延迟十分敏感的应用，如果和其他作业部署在同一台服务器上，由于不同作业对CPU、内存等资源的竞争，导致在线作业的访问延迟会明显增加，出现违反服务级别协议SLA(Service-Level Agreement)的情况，给企业带来难以承受的经济损失。亚马逊发现其在线业务的访问延迟每增加100ms就会导致销售额下降1%，对于每年销售额上千亿的亚马逊来说，销售额下降%1是巨大的损失[2]。因此企业出于保障在线作业QoS的考虑，往往为其配置独占服务器，设定远超其资源需求的CPU、内存等资源配额，这进一步导致了数据中心资源利用率的下降。在一份名为Quasar的研究中，研究人员对Twitter的一个生产环境的数据中心资源利用率做了30天的统计，结果图1-1，作业的平均CPU使用率为20%多，而为这些作业保留的CPU资源则占到总资源的70%多，作业的平均内存使用率为50%多，而为这些作业保留的内存资源则占到内存总资源的80%左右。



# 二、业界成功实践

随着越来越多公司开始意识到混部的巨大经济价值，也随着容器技术和新一代开源集群管理系统诸如Kubernetes、Mesos、Yarn的成长，目前国内外许多公司都在进行混部的研究和实践，诸如谷歌、百度、腾讯、阿里这样的公司成功把混部系统应用到了生产环境中，这些公司的混部系统能在保证在线作业服务质量受到影响比较小的情况下将离线作业和在线作业混合部署在同一个节点上。本节将会介绍这些混部系统的设计、实际效果以及优缺点对比。

## Borg

Borg[3]是谷歌内部使用的一套集群管理系统，通过资源预测、资源动态分配及动态扩缩容、任务打包调度、资源超卖以及有效的内核层面的资源隔离措施，使得Borg管理的集群有很高的资源利用率。

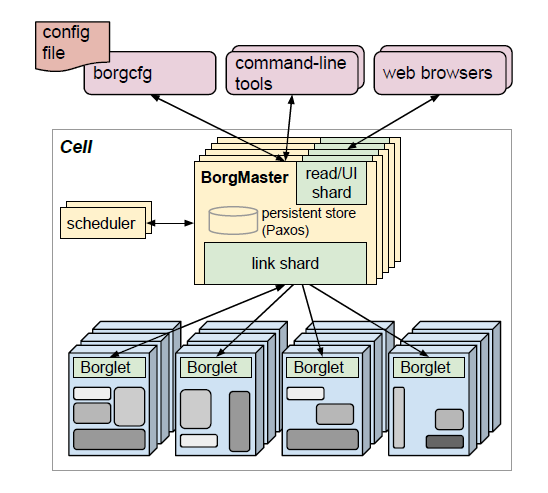


图2-1 Borg系统架构

图2-1是Borg的架构图，Borg集群管理系统主要由BorgMaster、调度器、基于Paxos的分布式数据库以及Borglet组成。

Borgmaster响应来自客户端的RPC访问请求，接受用户提交的作业或者返回集群的状态。同时Borgmaster还负责管理集群中所有的资源，包括节点、任务、alloc等。

调度器内部有一个记录了所有处于Pending状态待调度作业的优先队列，调度器周期性扫描这个队列调度作业，分可行性检测和打分两个阶段对作业进行调度。可行性检测是调度器筛选出满足任务约束条件和资源需求的节点，打分阶段通过一种E-PVM和Bestfit的综合打分模型进行打分，将作业调度到得分最高的节点上，当集群中的节点太多时，随机选出一部分比例的节点进行可行性检测和打分可以保证比较低的调度延迟，缺点是牺牲了更优解。当打分节点筛选出的节点没有足够的资源可以运行新任务时，Borg通过杀死更低优先级的任务为该任务抢占资源，被抢占资源的任务会重新进入等待调度队列。图2-2是Borg中作业的生命周期变化图。

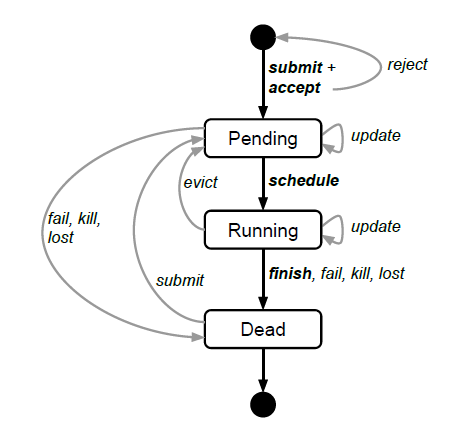


图2-2 Borg中作业和任务的状态变化图

基于Paxos协议的分布式数据库Chubby保存着集群的状态，同时Borgmaster也通过该分布式数据库提供的分布式锁进行选主，拿到锁的Borgmaster处于Active状态，当主Borgmaster挂掉时，其他的备份Borgmaster会重新选主，当挂掉的Borgmaster重新启动时，它会从Chubby中的最新的changelog恢复自己的状态。

Borglet是本地的Borg agent，主要有以下的功能：启动和停止任务；重新启动失败的任务；通过设置内核参数管理本地资源；提供debug日志查询；将本节点的状态汇报给Borgmaster和其他监控系统。

谷歌为了进一步降低Borg的调度延迟，提高其调度器吞吐量，做了如下三项优化：

* 1. 打分历史缓存
  2. 只对约束条件和资源限制相同的任务的一个实例进行可行性检查和打分
  3. 随机筛选一部分节点进行调度检查

Autopilot[4]是Borg的应用画像模块，主要作用是为作业确定资源配额以及实例数。Autopilot可以对作业进行纵向扩容，即为作业设置细粒度的CPU和内存使用配额，以减少作业的实际资源使用量和申请量之间的差额，同时确保作业不会出现资源不足的问题。图2-3是Autopilot的一个架构图，Autopilot中有三个闭环控制系统，一个用于HPA，两个分别用于作业CPU配额调整和内存配额调整。其VPA的推荐算法主要是滑动窗口算法和机器学习模型，通过这两个算法得出一个推荐值之后再乘以一个保险系数得到最终的资源推荐值。其HPA默认的算法也是基于滑动窗口的，根据当前任务的CPU利用率和目标CPU利用率推算出推荐实例数，或者也支持自定义的扩容函数。为了使得扩容更加平稳，其采取了如下的策略：缓慢缩容、忽略小的负载波动以及限制扩容的速率。

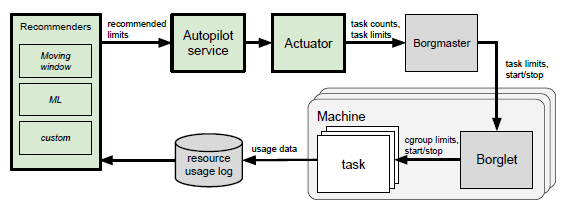


图2-3 Autopilot的数据流向图

为了保证在线作业的QoS，在线作业的内存配额是按照节点实际可用内存进行确定的，不支持超卖。Borg也在资源上做了区分，将资源分成可压缩的资源（比如CPU、磁盘带宽）和不可压缩资源（比如内存、磁盘），当节点的可压缩资源不足时，Borg限制离线作业的可用资源量，当节点的不可压缩资源不足时，Borg杀死离线作业释放资源。为了隔绝在线作业之间的干扰，Borg对在线作业采取的策略时每个在线作业绑定物理核，而离线作业可以在任何核上运行，但是离线作业分配到的CPU份额比较少。

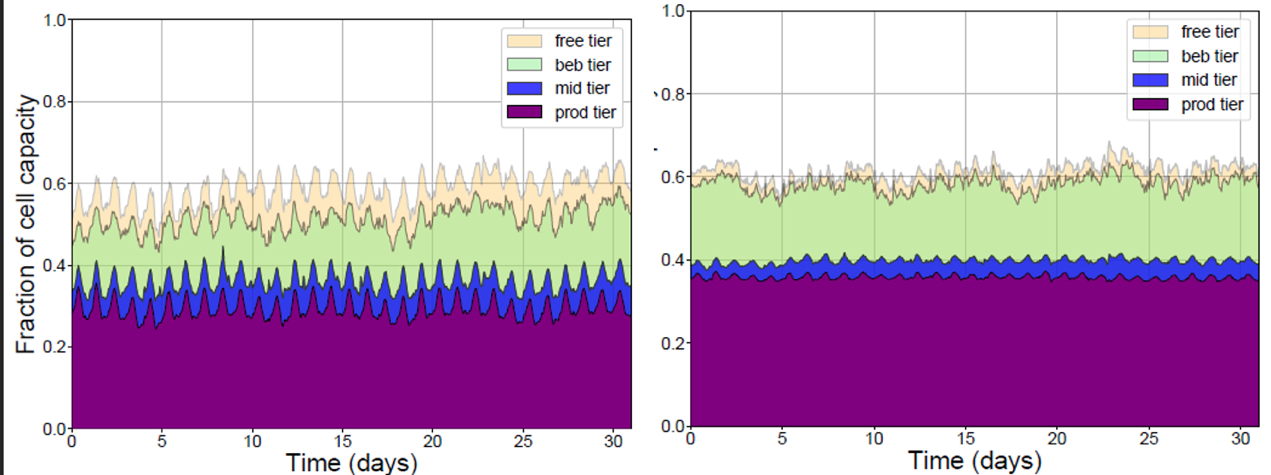


图2-4 Borg多个集群的平均资源利用率

图2-4展示了Borg管理集群的资源利用率[5]，其中CPU和内存平均利用率均达到了60%左右，充分说明了Borg在提高集群利用率方面非常成功。

## Caelus

腾讯的场景中主要有两大类四小类作业：

在线作业：（1）容器化作业，基于k8s部署。（2）非容器化作业，主要是不适合容器化的作业，对性能影响较大，如存储作业ceph。

离线作业：（1）容器化部署在k8s的应用k8，如自研k8s离线应用，或spark on k8s等。（2）Hadoop类等非容器化作业。

1. Caelus架构

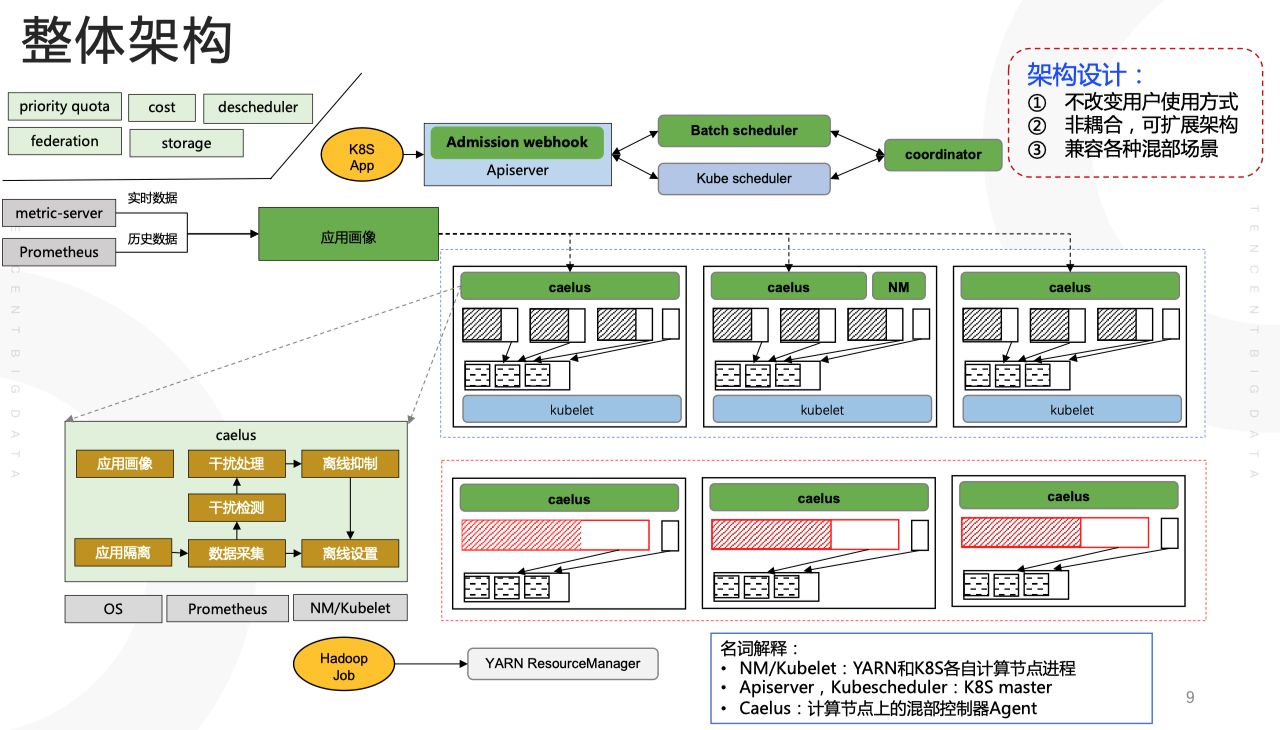


图2-5 Caelus架构图

腾讯Caelus的整体架构如图2-5所示，主要由数据体系，应用画像，Agent，调度器四个模块组成。

Agent分别由Kubelet(k8s agent), Caelus(Caelus Agent), NM(yarn agent)三个组件组成。

调度器分别由kube-scheduler, batch-scheduler, RM分别管理k8s在线作业，k8s离线作业，以及Hadoop作业的调度。

1. 应用画像

Caelus兼容两种预测方式：

（1）远程预测：使用vpa-recommender算法；全局考虑应用下所有pod的资源使用情况，统一进行资源预测；新扩容pod根据历史数据立即获取到资源资源。

（2）本地预测：同样使用vpa- recommender算法；根本节点资源实际使用量预测，预测包含在线作业与系统进程；在线作业资源突变时，可快速作出反应。

1. 混部Agent

混部Agent以k8s daemonset的形式部署在集群中，采用去中心化的设计，在本地做资源画像，反馈到上层决策，又管理离线大框，利用TencentOS以及Linux原生的特性做内核隔离，以保证节点上的作业稳定。

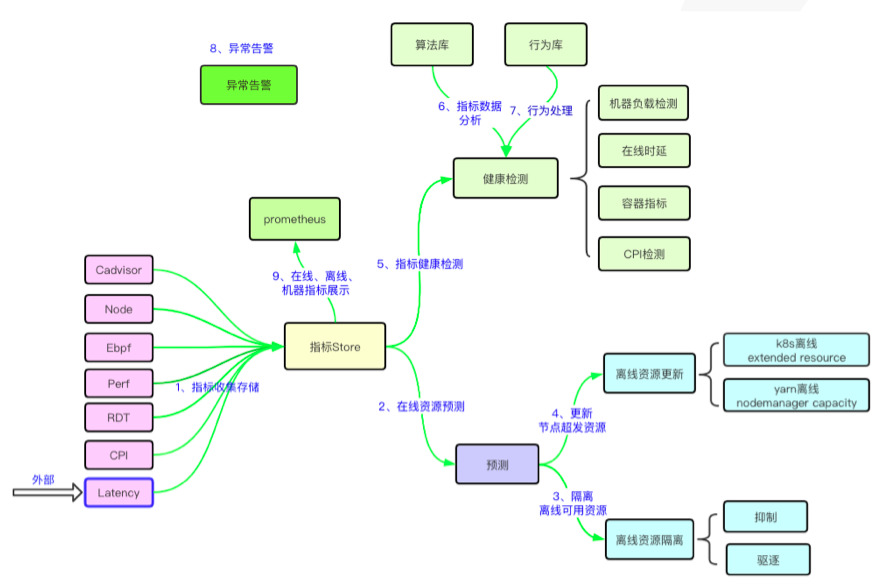


图2-6 Caelus Agent功能

因为资源隔离是混部的核心，所以腾讯做了大量资源隔离相关的工作，Caelus号称全维度的资源管理。如图2-6所示，管理的维度包含磁盘IO，硬盘空间，Inodes，进程状态数，内存，CPU，网络出入带宽。更底层的还有L3 cache，内存带宽的隔离。并且采用零入侵k8s原逻辑的方式，在与k8s besteffort, burstable cgroup同级的目录下创建offline cgroup，将离线作业都放入该cgroup中，采用离线大框管理所有离线作业共享的可用资源。

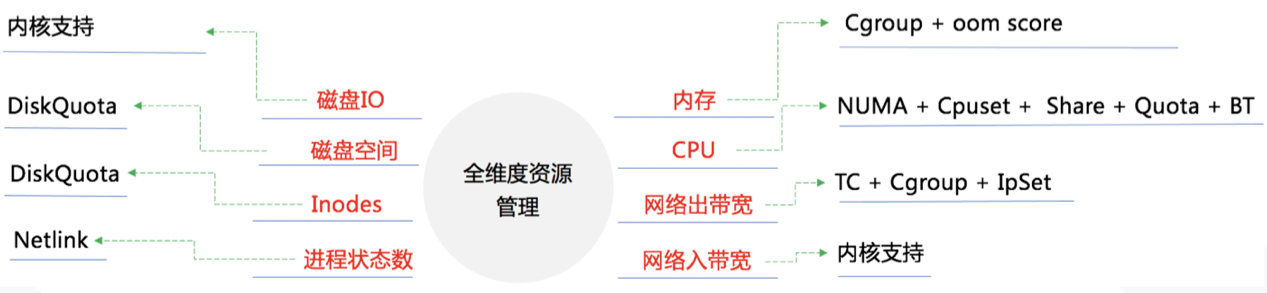


图2-7 Caelus 资源隔离

1. 调度器

腾讯Caelus的调度方案是将k8s kube-scheduler, batch-scheduler, yarn三个调度器结合，顶层使用coordinator协调三个调度器之间的资源。使用batch-scheduler的目的是解决很多离线作业的最小同时执行任务数量约束，而腾讯Caelus基于社区batch-scheduler进行改造，在支持gang-scheduling的同时，将每个job的pod同样看待，一次计算分数，而非kube-scheduler对每个pod都计算一次调度分数，极高地提高了调度性能。另外，为了将yarn的调度框架接入k8s中，Caelus将nm以pod的形式部署在集群中，动态更改pod所能用的资源，就能动态更改yarn所能用的资源。但是为了降低复杂性，某个节点的离线作业要么由yarn托管，要么由batch-scheduler托管，不能同时由两个调度框架托管某节点的离线作业。

1. 混部效果

业务方为AI训练任务的集群，通过混部后，总体使用率提升了15%，但原因是受磁盘空间限制，若挂远程盘，使用率可提升到60%。

# 三、关键技术

本节从应用画像、作业调度、节点Agent以及数据中心四个方面对在离线统一资源调度的关键技术进行研究。

## 应用画像

应用画像模块是在离线统一调度的重要支撑，其主要功能包括资源预测、资源配额以及应用亲和性分析。

1. 资源预测

资源预测指的是根据作业的历史运行数据预测未来一段时间作业的资源使用情况。资源预测的结果可以指导作业的水平扩缩容和任务的纵向扩缩容以及设置离线业务的可用资源量。可用于资源预测的算法有：ARIMA自回归积分滑动平均模型、LSTM长短期记忆模型、GRU门控循环单元、滑动窗口指数衰减。

ARIMA 模型[6]是在平稳的时间序列基础上建立起来的，因此时间序列的平稳性是建模的重要前提。检验时间序列模型平稳的方法一般采用 ADF 单位根检验模型去检验。当然如果时间序列不稳定，也可以通过一些操作去使得时间序列稳定（比如取对数，差分），然后进行 ARIMA 模型预测，得到稳定的时间序列的预测结果，然后对预测结果进行之前使序列稳定的操作的逆操作（取指数，差分的逆操作），就可以得到原始数据的预测结果。

LSTM[7]最早用于自然语言处理，由于其出色的长期记忆能力，LSTM非常适合用于进行资源预测。典型的LSTM神经网络单元主要由三个门构成：输入门、遗忘门和输出门。输入门获取外部的输入数据并进行处理，记忆单元输入门获取上一次LSTM神经网络单元的输出，遗忘门决定什么时候去忘记输出的结果。输出门获取所有计算结果并且生成LSTM神经网络单元的输出。在自然语言处理中，通常会将LSTM神经网络单元的输出经过softmax激活函数的处理。在资源预测模型中，LSTM神经网络单元的输出层之后还会加上一个线性回归层。

假设输入的时间序列是，记忆单元的状态是，输出的时间序列是，则LSTM的计算步骤如下：

其中W是不同节点的连接参数，b是偏移量。记忆单元的隐藏状态计算如下：

其中表示sigmoid激活函数，\*表示两个向量或者矩阵的标量乘积，g和h是标准sigmoid函数的变形，域值分别为[-2,2]和[-1,1]。

损失函数选取平方差损失函数：

其中y表示真实资源使用值，pt表示资源预测值。为了减小训练偏差同时避免局部最小值，用自适应学习率的Adam优化器进行反向传播计算。

GRU门控循环单元[8]和LSTM类似，但是在计算和实现上更简单。一个典型的GRU单元由如下两个门所组成：复位门r和更新门z。和LSTM神经网络单元相似，t时刻记忆单元隐藏层的状态由t-1时刻隐藏门的状态和t时刻的输入计算得到，表达式如下：

滑动窗口配合指数衰减是VPA recommender[9]所采用的算法。假设滑动窗口大小为N天，则资源预测模块会利用这N天的资源使用历史数据生成应用的资源推荐值。而指数衰减则是越老的数据权重越小，越新的数据权重越大，权重计算公式如下：

其中τ是半衰期，是当前时间，是该数据点的采样时间。于是直接根据历史数据获得的预测值可以通过下式计算得到：

这几种预测算法的准确度从高到低排序依次是：GRU、LSTM、ARIMA、滑动窗口加指数衰减，计算开销从低到高依次是：滑动窗口、ARIMA、GRU、LSTM。虽然GRU和LSTM的预测比ARIMA更加准确，但是差别并不大，滑动窗口加指数衰减的做法在Borg和K8S的VPA中都有应用，经过大规模生产验证，是一种可靠的预测算法。图3-1和3-2分别是LSTM、GRU和ARIMA预测准确度的对比以及滑动窗口加指数衰减和LSTM预测准确度的对比，其中图3-1来自[10]，图3-2来自[11]。

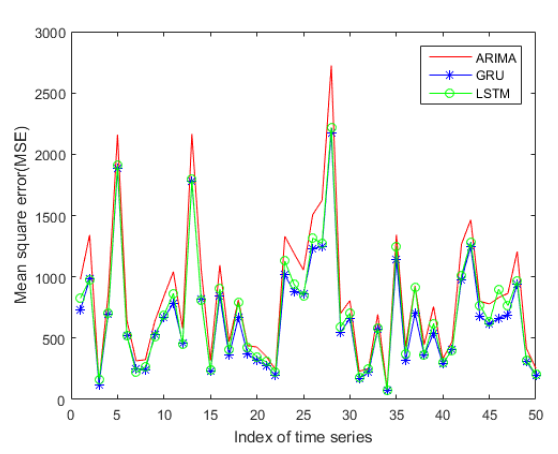


图3-1 LSTM、GRU以及ARIMA预测的错误值

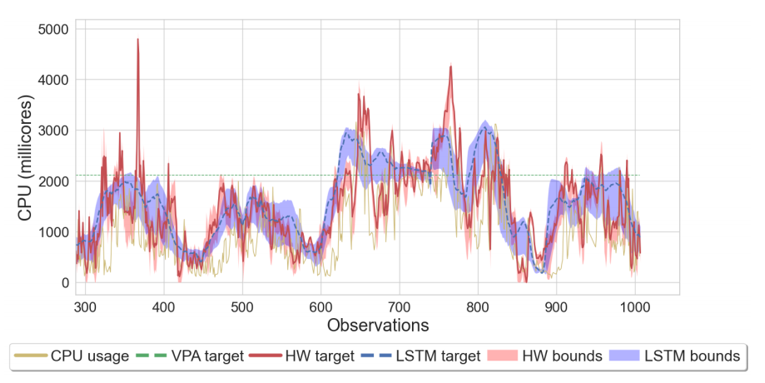


图3-2 VPA、LSTM预测目标值和上界值对比

1. 资源配额

资源配额是指根据Task的历史数据和Task本身的负载特性确定合理的资源划分。相关工作中动态划分的资源包括CPU、LLC容量、LLC路数、L2与L3的带宽、内存容量、磁盘带宽以及网络带宽等。常见的做法有：用户根据经验指定各个资源申请量、固定资源配比以及构建压测资源间表达式，依据测试数据配置资源配额。

用户根据经验指定各个资源的申请量是指由提交作业的用户在作业声明文件中指定各个资源的申请值，这种做法需要用户预先对作业的各种资源使用情况进行测试，然后给出一个可以满足作业资源要求的值。人工确定应用资源的缺点包括：1、如果应用的资源使用量是周期性波动的，人工确定应用资源需求是按照应用的峰值资源需要进行分配的，在波谷时应用的资源使用量是远低于峰值资源需求的，导致了极大的资源浪费，也限制了应用可以调度到的节点数量；2、有数据表明人工确定内存资源使用量会比各种自动化方式有更高的几率出现oom的情况；3、随着应用的不断升级，每次升级都需要人工重新测试应用的资源使用情况，消耗了额外的人力，也导致应用的升级周期延长。

固定资源配比是指比如系统关注的主要指标是CPU和内存的利用率，通过固定CPU/内存的比值来确定资源的分配方式。这种做法一般可以和DRF[12]的调度算法结合，实现绝对的资源公平。这种做法存在的问题包括固定资源配比关注的资源仅为CPU和内存，没有考虑到其他资源的利用率问题。

构建压测资源间表达式，依据测试数据配置资源配额需要在调度前将应用在测试环境中进行运行。文献PARTIES[13]中提到应用存在的性能拐点问题，当应用的rps到达拐点时，应用的99%访问比例延迟将会变为原来的好几倍，于是应用出现明显性能拐点时的rps就可以作为该应用的最大负载，并将最大负载时应用的99%访问比例的延迟作为应用的QoS标准。有了应用QoS的标准之后，就可以根据不同资源分配情况下应用的访问延迟，判断应用是否达到了QoS的要求，当应用的QoS不达标时就需要增加应用的资源配额，如果应用的QoS达标了，PARTIES会主动减少应用的某种资源的配额，如果其QoS达标，说明此时应用不需要用到那么多的资源，从而减少应用分配的资源和使用资源的差值。

确定应用的资源配额和应用的实例数之间存在一个矛盾，减少单实例的配额时会使得满足QoS需要的实例数会增加，但是实例数的增加会给cpu、内存带来额外的压力，因为应用只要运行就存在一定的基础开销，增加单实例的配额会减少所需要的实例数，但是可能会存在资源碎片问题，总之需要在资源配额和实例数之间进行一个权衡。

1. 应用亲和性

Task亲和性：有的Task有节点亲和性以及Task之间亲和性和反亲和性的要求。其目的是辅助调度决策，尽量使不同资源亲和性的应用调度到一起，使得各种资源成物理机容量配比的方式进行消耗，减少应用间产生资源竞争和性能干扰的情况。常用的做法是将应用按资源使用情况分成不同的类别，比如CPU密集型、内存带宽密集型、IO密集型、网络带宽密集型等，再使用分类算法根据应用运行的历史性能数据进行分类。常用的分类算法包括聚类算法K-Means、机器学习方法SVD协同过滤等。

k-means[14]的基本思想是设置一个目标距离的计算函数，两个目标的距离越近，相似度越高。文献HySARC[15]中介绍了一种利用k-means对作业和节点进行分类的方法，其调度的基本思想是将作业按照资源需求分成k类，将节点按照可提供的资源分成k类，每一类作业固定调度到一类节点上，这样作业调度就变成了在每一类中按照短作业优先或者最早结束时间有限的方式调度作业。这种做法能够在较好的解决应用亲和性的问题，但是需要处理好聚类阶段的开销。

SVD协同过滤[16]是一种推荐算法，根据用户对于某些商品的历史评分，预测用户对于另一些商品的评分。文献Paragon[17]中介绍了一种利用SVD预测应用在不同资源配置情况下在不同硬件服务器上所能达到的性能以及其在混部情况下的受到的性能干扰，最后通过贪心算法选择性能干扰最小并且满足应用QoS要求的节点和资源配置。这种做法的好处是充分考虑了应用的亲和性问题，但是会有比较大的计算开销。

## 作业调度

作业调度指的是根据当前各节点的资源使用情况以及待调度作业的资源需求，在满足作业QoS或者吞吐率的目标下，将作业调度到满足资源需求且性能干扰最小的节点上。一个集群管理系统的调度效率以及调度效果会受到调度器架构和调度算法的综合影响，本小节首先会介绍这两个方面。随着集群管理系统的不断发展，开源社区涌现出了Mesos[18]、Yarn[19]、Kubernetes[20]等许多优秀的集群管理系统，这些管理系统有的只是针对在线作业场景设计的，缺乏对批处理调度的支持，有的是从大数据计算框架衍生出来的，本身就不适合做在线业务的管理，只适合批处理业务的管理，Kubernetes虽然有一统在离线调度的发展趋势，但是目前迁移到k8s上的批处理计算框架仍然不多，目前许多企业的大数据管理平台仍然是yarn，在在线作业集群资源利用率偏低的情况下，如何将yarn和Kubernetes管理平台协同起来，统一调度在离线业务是一个有现实需要的问题。 同时，虽然Kubernetes支持多调度器实现针对不同负载类型的调度策略，但是多调度器不可避免地带来了资源竞争的问题，如何协调多调度器的资源竞争也是一个有现实需要的问题。

1. 调度器架构

调度器架构可以分成集中式调度器、分布式调度器、分层调度器、共享状态调度器以及混合架构调度器。

集中式调度器只有一个活跃的调度器实例，使用一种集中化的调度算法调度所有作业。集中式调度器的优点是可以充分使用集群的全部资源调度作业，能够支撑全局最优的调度算法的执行，并且可以实现基于优先级的资源抢占。缺点是集中式调度器扩展性差、并发度低，调度一个作业需要的时间长，因此更适用于节点数中等或者比较少的集群，同时使用一种算法支持不同的调度策略难度很大。使用集中式调度器的集群管理系统有Borg、YARN。

分布式调度器又称为静态划分集群调度器，它是指静态将一整个集群划分成不同的部分，每个部分有一个负责的调度器，这样做的好处是减轻了每个调度器的负载，加速了调度过程，但是由于每个调度器只负责自己管理节点作业的调度，导致了集群范围内的资源不能被充分利用，很有可能出现的一种情况就是某个调度器分配到过多的作业，资源不足导致作业不能调度，而其他调度器分配到的作业少有空闲资源可用却不能把这部分资源用于运行其他调度器分配到的作业。

分层调度器是为了解决静态划分集群调度器的缺点而出现的。有的分层调度器通过一个中央协调器划分每个调度器可用的资源量，比如Mesos，也有的分层调度器通过一个中央调度器调度作业到局部调度器，每个局部调度器负责将作业的任务调度到节点上，比如Mage[21]。分层调度器的优点是调度效率高，能一定程度解决静态划分集群导致不能充分利用集群中资源的问题。

共享状态的调度器为了提供高可用性和扩展性，将除共享状态之外的功能剥离出来成为独立的服务。这种调度器为了解决共享状态变更的并发安全问题以及解决悲观并发的低并发问题，引入了乐观并发的思想，不同于悲观并发只支持对共享变量的串行访问，乐观并发假设其他线程不回去修改共享变量，只在更新共享变量的值时进行校验。共享状态的调度器优点是并发性高、扩展性好，缺点是如果经常出现多个调度器资源冲突的情况，导致调度效率不如采用悲观并发的调度器。

混合架构的调度针对长短作业的本身特性而采用不同的调度器，其使用中央调度器调度长作业，使用分层调度器调度短作业。这种混合架构的调度器在面对在线业务和离线业务的场景下都能实现很好的调度效果。

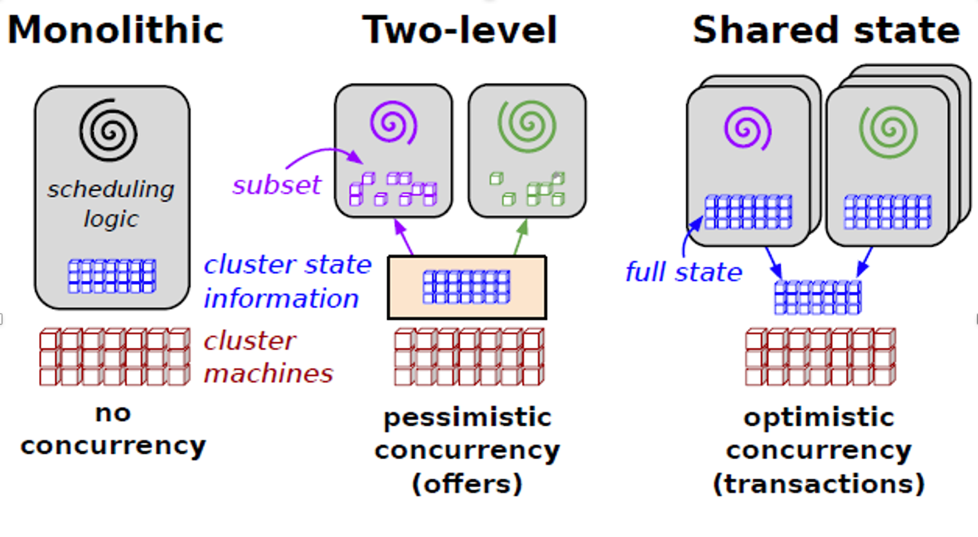


图3-3 调度器架构类型

1. 调度算法

为了满足作业调度中的资源公平性、作业资源需求、集群负载均衡、集群吞吐率等多个优化目标，相关工作了提出了许多不同的算法。

启发式算法：基于专家直观经验构造算法用于寻找最有调度结果，启发式算法由于简单直观，易于理解和修改，被许多在离线混部调度算法所采用。文献Spread-n-Share[22]中设计了一种综合考虑作业优先级、作业扩容系数和作业各种资源需求的调度算法，调度器根据以上的考虑点筛选合适的节点调度作业。调度器周期性扫描带调度作业的优先队列，取出队首作业进行调度，调度时会尽量满足扩容系数的要求，扩容系数表示了作业在是否需要调度到更多节点上的偏好，调度系数等于1的作业会尽可能将副本调度到尽可能少的节点上，调度系数为k的作业会尽可能将作业调度到接近N/k个节点数上。调度器将节点按其空闲核数进行分组，优先考虑将作业调度到属于同一个组内，当某一个组内的节点数不足以调度作业时，SNS搜索所有的集群节点调度作业。不管是哪种情况，SNS都会筛选出k个最空闲的节点，这么做的目的是让节点之间负载均衡以及较少作业副本间的性能干扰。节点的空闲度由公式算得，其中是节点已经使用的核数，是节点上已经使用的内存带宽，是已经分配的LLC的路数。当调度器筛选出足够的节点后就将作业调度到对应节点上运行，如果没有筛选出足够的节点数，调度器将会考虑次优的扩容系数，当没有次优的扩容系数可选时，调度失败，作业被重新放回队列过一段时间后再被调度。图3-2是其调度的流程图。启发式算法的局限性在于:1、当需要优化的目标较多时,调度问题的复杂度上升,设计兼顾多个目标的启发式算法具有一定的难度;2、启发式算法难以保证算法给出的解是全局最优解。

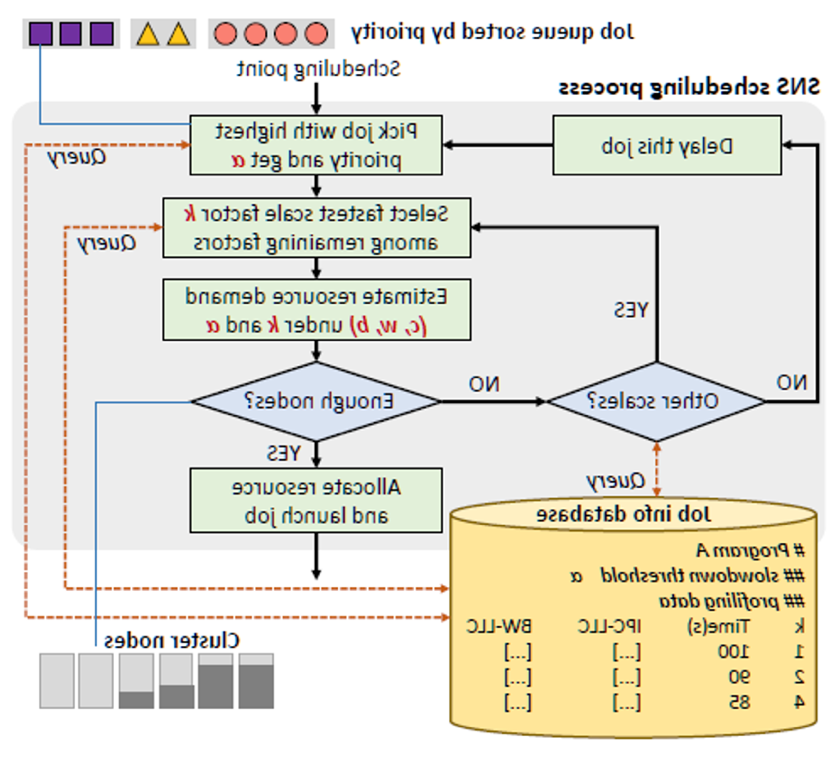
**

图3-4 SNS调度流程

打分算法：当调度优化目标较多时,对每个目标设定量化规则,并使用这些规则对每个节点进行打分,将最终的加权分数作为节点最终得分,最后将作业调度至得分最高的节点。文献[23]介绍了一种统筹考虑高低优先级作业混搭、不同资源密集型作业混搭、数据本地性、负载变化规律匹配的打分方法，其中每一个优化目标有一个得分值，其值为0或1，比如如果节点上已经有CPU密集型作业了，如果带调度作业也是CPU密集型，则该节点在此轮调度的不同资源密集型作业混搭目标上得分为0，同时不同的目标重要性不同，高低优先级作业混搭关系到在线作业的服务质量保证，最重要，因此要给该目标设置最大的权重，负载变化规律匹配的重要性最小，给该目标设置最小的权重，实验结果显示该打分规则能够较好的完成混部的作业匹配任务。打分算法的优点是易于扩展，计算开销低，其局限性在于：1、每个规则的权重作为调度算法的超参数，对调度结果影响较大，寻找最佳的超参数设置需要经过不断的调试；2、基于打分规则的调度算法同样难以保证算法给出的解是全局最优解。

整数线性规划：作业调度问题可以抽象为组合优化问题,而整数线性规划问题(integer linear program,简称ILP)作为组合优化的一种解决方法,可用于解决作业调度问题。文献Medea[24]介绍了一种基于ILP的调度算法，主要考虑满足如下的调度目标：1、调度长作业的所有副本；2、满足长作业的约束条件；3、考虑到每个节点的资源使用情况；4、满足如下的目标：最小化资源碎片、节点间负载均衡以及最小化使用的节点数。

表1 Medea的优化函数中各个参数的含义

|  |  |
| --- | --- |
| 符号 | 含义 |
| k | 需要调度的长作业数量 |
| N | 集群中的节点数 |
|  | 长作业i的副本数 |
|  | 节点n的空闲/已使用资源量 |
| m | 约束条件的个数 |
|  | 某部分值在目标函数中的权重 |
|  | 足够大的整数，跟节点有关 |
|  | 如果长作业i所有的副本都已经被调度，其值为1，否则为0 |
|  | 如果长作业i的第j个副本调度到了节点n上，其值为1，否则为0 |
|  | 长作业i的第j个副本资源需求 |
|  | 最小资源需求 |
|  | 最少和最多违反约束条件 |
|  | 违反约束条件 |
|  | 如果调度结束之后节点的剩余资源大于等于最小资源限制，则为1，否则为0 |

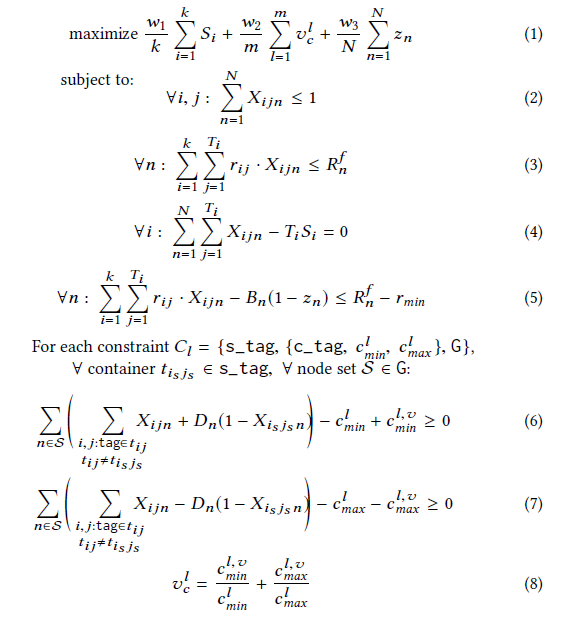


图3-5 Medea的ILP公式

图3-3中是Medea用到的整数线性规划公式。公式1的优化目标是1、尽可能一次调度所有k个长作业；2、最小化违反约束条件的个数；3、通过最小化剩余资源很少的节点数减少资源碎片（公式5）。公式2-4则是长作业的所有副本都被调度且节点满足作业资源需求的等式。公式8则是长作业l违反约束条件量化表示，某个长作业违反约束条件的成都等于其违反约束条件的最小值占所有长作业作业违反约束条件最小值的比重加上其违反约束条件的最大值占所有长作业作业违反约束条件最大值的比重，之后再通过CPLEX优化器进行求解。整数线性规划的调度算法的局限性在于:(a) 算法开销较大,文献Medea通过实验证明,在同等规模下,整数线性规划算法的求解时间开销均高于启发式算法和基于打分规则的调度算法;(b) 目标函数的构造对调度质量至关重要,对多个优化目标进行量化表示具有一定的难度。

1. 不同集群管理系统调度协同

由于集群管理系统的不断发展，一个企业内部的不同类型的负载可能运行在不同的集群管理系统之上，运行在线业务的集群面临资源利用率过低的问题，运行离线业务的集群则面临资源不够的问题，那么为什么不将离线作业迁移到在线集群上或者将在线作业迁移到离线集群上呢？将在线作业迁移到离线集群可能会遇到的问题就是离线集群管理系统没有对在线作业调度优化的算法，没有提供协作任务之间通讯的组件等等问题，导致在线作业迁移到离线集群面临许多困难。同样，离线作业迁移到在线作业管理集群也会遇到在线作业集群管理系统调度吞吐量小，对应的计算框架没有迁移到在线作业管理集群等问题，也是面临许多困难。虽然随着集群管理系统的发展，使用一套集群管理系统同时管理在线作业和离线作业是趋势，但是短时间内一个集群内存在多套集群管理系统的现象也是存在的。目前多个集群管理系统协同调度的方式主要有：弹性设置离线资源可用量、引入资源协调层以及使用通用资源调度程序。

弹性设置离线资源可用量：根据在线作业的资源使用情况确定离线的可用资源。具体的做法有两种：1、动态划分在离线节点；2、动态调节离线agent可用资源量。字节跳动的混部实践[25]采用的做法就是动态划分在离线节点，将CPU/内存/GPU使用率或者在线作业的QPS作为在线作业的负载指标，当在线作业负载较低时，将低负载节点上的在线作业迁移到其他节点上，此时节点处于在离线的中间状态，不会有作业调度到该节点上，当节点上的作业迁移完成后，由错峰控制平面杀死节点上的在线作业，完成之后节点转变为离线节点，离线作业可以调度，当在线作业负载增高时，由错峰控制平面杀死离线作业，之后节点变为在线节点，在线作业可以调度到该节点上。腾讯的做法是动态调节离线agent的可用资源量，由应用画像模块预测出下一段时间节点的在线作业资源使用情况之后，通过如下的公式计算出节点上离线作业可用的资源量：

节点离线作业的可用资源量=节点总共资源量-在线作业使用资源量-系统组件使用资源量-buffer大小

动态调节离线agent可用资源量比动态划分在离线节点能实现更精细的资源划分，但在实现上会更加复杂。

引入资源协调层的做法是引入一个资源管理器动态调节多个集群调度框架之间的可用资源量。阿里的sigma&fuxi[26]采用的是这种做法，好处是可以在一定程度上解决多管理框架的协同调度问题，缺点包括：1、0层的实现较为复杂；2、需要对哥哥管理系统做一定的修改以支持0层的资源协调作用。

使用通用的资源调度的程序做法是由各种调度框架的shim将不同的调度框架接入到调度模块。Apache的Yunikorn[27]为混合工作负载提供统一的跨平台调度体验，目前支持Kubernetes，下一步计划支持yarn。使用通用资源调度程序的好处是可以感知到全局的资源使用情况，可以Yarn的支持；2、Yunikorn内置的调度器没有经过大规模生产环境的验证，有可能存在调度性能的问题。

## 节点agent

节点Agent主要是每个节点都会有一个Agent进程，根据调度器下发的信息，做相应资源动态管理的划分，以及数据采集，干扰检测与干扰处理等工作。

1. 数据采集

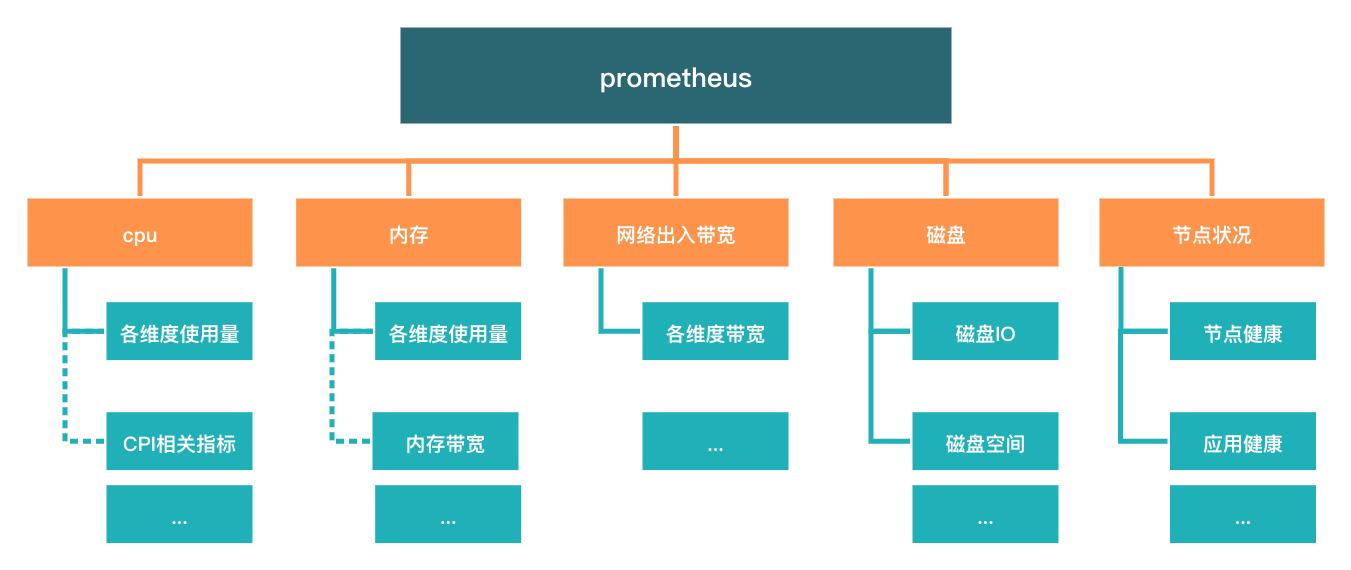


图3-6 数据采集指标

如Kubernetes中的kubelet，kubelet内置了cAdvisor采集容器资源使用情况，如容器CPU、内存、网络、文件系统等的资源占用情况。metrics-server 提供整个集群的资源监控数据。

但是kubelet采集的数据可能是不足够的，因为通常混部系统还需要其他信息用作分析。如Google的CPI2[28]使用perf\_event工具采集节点上的CPU\_CLK\_UNHANLTED.REF, INSTRUCTIONS\_RETIRED 两个数据指标，用作CPI的数据指标，建立性能干扰检测模型。PARTIES[13]采集了应用的内存带宽，LLC带宽等数据，验证实验中各应用对各指标的敏感程度。Google的Bubble-up论文[29]中采集了内存带宽数据，构建出了论文中的pressure score。

1. 资源隔离

在离线混部会在一台机器上同时运行着时延敏感的在线作业，还有计算密集型的离线作业，意味着一台机器上的各项资源会共享给该机器上运行的所有应用，应用之间就会竞争共享资源，从而引发性能干扰问题。由于在线作业对时延十分敏感，而CPU、内存带宽等资源上的性能干扰会严重影响在线作业的服务质量，所以资源隔离是混部中关键的技术点。目前已有的资源隔离技术能做到使用量上的隔离，比如cpu时间片，内存容量等，尽管这些隔离手段还是无法消除底层的一些性能干扰问题，比如LLC被冲，TLB刷新，内存带宽等问题。不过依然可以使用现有的一些技术，降低这些底层性能问题的影响。表2展示了目前可行的资源隔离方法。

表2 共享资源与隔离技术汇总

|  |  |
| --- | --- |
| 共享资源 | 隔离工具 |
| 超线程  CPU | (1)Linux cpuset为应用分配特定的物理核。  (2)Linux quota, shares 限制应用时间片 |
| Power | acpid调频工具，调整cpu频率 |
| LLC 容量  LLC 带宽  Memory 带宽 | (1)CAT(Cache Allocation Technology)技术可用来做cache way partition  (2)MBA(memory bandwidth allocation)提供了作业级别的L2 cache到L3 cache 带宽的划分  (3)Cache的访问带宽会间接影响访存带宽。 |
| Memory 带宽 | NUMA 给CPU分配特点的内存节点，可影响内存带宽，但也并非直接限制内存带宽。 |
| Memory 容量 | Linux memory cgroups 可以限制容器或pod级别的内存容量 |
| Disk 带宽 | Linux blkio cgroups 可用来限制应用的硬盘带宽 |
| Network 带宽 | Linux qdisc 令牌桶机制控制外出流量带宽 |

1. 离线大框

离线大框并非一个具体技术，而是把给离线作业划分的所有资源统称为离线大框，所有离线作业共享该离线大框里的资源，混部系统能够动态调整离线大框的资源量，从而影响离线作业可用的资源量。

在系统中，统一地将离线作业聚合到离线大框，有利于离线作业整体资源的调整。百度Matrix[30]、腾讯Caelus[31]都在机器上创建offline cgroup，用于管理离线作业共享的cpu、内存等资源。

图3-7 离线大框cgroups

1. 干扰检测及干扰处理

如在小节3.3.2资源隔离中提到的资源竞争引起的性能干扰，当性能干扰产生时，在线作业的服务质量就会收到影响，虽然性能干扰一定程度上可以在调度阶段，通过历史数据及预测数据等进行预估，但是多个应用部署到一台物理机上，必定会产生资源竞争，若仅依靠上层调度器发现服务质量下降，随后进行应用迁移，重调度等方式减小物理机资源竞争，可能已经为时过晚，具有一定的滞后性。所以通过本地agent对性能干扰进行检测，效率会更高，更有效地避免在线作业的服务质量受到影响。但是仍然可以通过监控数据，检测节点上在线作业是否受到性能干扰，并且在性能干扰发生了，如何识别干扰者，以及处理干扰者。

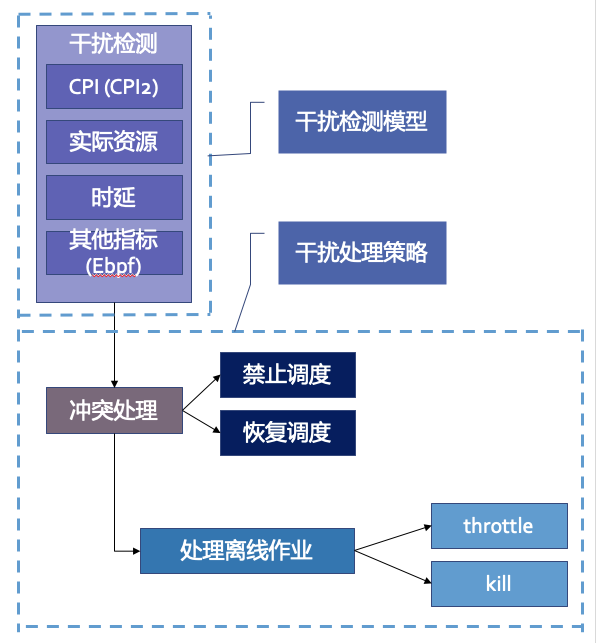


图3-8 干扰检测模型与干扰处理策略

干扰检测

干扰检测的关键是构建干扰检测模型，干扰检测模型通常涉及作业当前负载load，作业相关硬件指标hw。在离线作业有不同的性能指标，例如，在线作业的性能指标可能是请求的响应时间(Response Time)，作业一段时间内的失败率；而离线作业的性能指标通常是作业的吞吐率等[32]。RT、吞吐率这些性能指标确实能真实地反映应用的运行状态，但是在很多场景下，应用的RT、吞吐率属于企业敏感数据，这些数据对于干扰检测模型是黑盒的，除非用户主动输入。现有的一些相关工作成果发现IPC(instructions per cycle), CPI(cycles per instruction), MIPS(million instructions per second)等非业务敏感的硬件指标也可以充当应用性能指标的评估。

CPI2[28]发现了应用的CPI指标与应用的表现存在明显的相关性，所以该论文将CPI作为应用的性能指标，依据应用当前的性能指标及历史指标建模，以识别当前是否存在在线应用受到性能干扰，在检测到在线应用受到性能干扰后，根据干扰模型识别出干扰者，对干扰者进行压制，以尝试恢复受干扰者的性能。国内的混部系统如阿里巴巴Fuxi&Sigma[33], 腾讯Caelus， 百度Matrix都公开表示引用过该方法，一定程度上说明了该论文模型方法的可靠性。CPI2干扰检测主要存在的缺陷是该指标仅与CPU有关，若应用是I/O密集型的，CPI2的方法模型效果较差。

BubbleFlux[34]则使用应用的IPC指标作为应用的性能表现，该指标与CPI相差不大，BubbleFlux根据在线作业的IPC值，动态调整在线作业与离线作业的CPU时间比例，以恢复在线应用的服务质量，同CPI2一样，BubbleFlux同样存在仅能影响CPU资源的缺陷。

干扰处理

干扰处理是检测到了节点上发生性能干扰了，对干扰者或者直接对离线作业进行处理，处理方式大多数系统做法一致，Google Borg，百度Matrix，阿里巴巴Fuxi&Sigma，腾讯Caelus都是先调整离线作业的资源，其中如果调整后，在线作业还无法改善，会直接kill离线作业，以及驱逐离线作业。

## 数据体系

数据体系是做混部的基石，混部系统必须数据体系做调度决策，应用扩缩容，资源动态调整等工作。所以数据体系除了存储社区原生工具采集的数据外，还需要存储应用画像的资源预测、应用资源配额等数据。

Kubernetes 社区提供了一些列的工具来监控容器和集群的状态，并借助 Prometheus 提供告警的功能。

* cAdvisor 负责单节点内部的容器和节点资源使用统计，内置在 Kubelet 内部，并通过 Kubelet /metrics/cadvisor 对外提供 API
* [InfluxDB](https://www.influxdata.com/time-series-platform/influxdb/) 是一个开源分布式时序、事件和指标数据库；而 [Grafana](http://grafana.org/) 则是 InfluxDB 的 Dashboard，提供了强大的图表展示功能。它们常被组合使用展示图表化的监控数据。
* [metrics-server](file:////setup/addon-list/metrics) 提供了整个集群的资源监控数据，但要注意
  + Metrics API 只可以查询当前的度量数据，并不保存历史数据
  + Metrics API URI 为 /apis/metrics.k8s.io/，在 [k8s.io/metrics](https://github.com/kubernetes/metrics) 维护
  + 必须部署 metrics-server 才能使用该 API，metrics-server 通过调用 Kubelet Summary API 获取数据
* [kube-state-metrics](https://github.com/kubernetes/kube-state-metrics) 提供了 Kubernetes 资源对象（如 DaemonSet、Deployments 等）的度量。
* [Prometheus](https://prometheus.io/) 是另外一个监控和时间序列数据库，还提供了告警的功能。
* [Node Problem Detector](https://github.com/kubernetes/node-problem-detector) 监测 Node 本身的硬件、内核或者运行时等问题。

# 四、挑战

内存带宽

内存带宽对应用的性能影响很大，但目前并无硬件或软件的手段可以直接限制应用的内存带宽，通常是通过结合限制应用的其他资源，从而限制应用的内存带宽。阿里巴巴LIBRA提出的动态内存带宽的方法，验证通过动态调整应用内存带宽，与现有的内存QoS策略相比，能够使混部场景下的离线作业性能增加52.8%。包括阿里之前Fuxi&Sigma中提到的方法是结合 MBM(memory bandwidth monitoring)与cfs bandwidth control间接限制应用的内存带宽。腾讯Caelus是通过CAT(Cache Allocation Technology)+NUMA间接限制应用的内存带宽。另外还有intel的MBA(Memory bandwidth allocation)限制l2-l3的cache带宽，也可以间接限制应用的内存带宽。解决内存带宽问题是混部系统中很重要的一个方向。

集群多调度系统的统一

k8s的调度系统原生适合支持长作业应用，但是很多离线作业往往是短作业应用，并且考虑到历史问题，以及其他调度框架如yarn/mesos在离线作业管理方面的积累，很多企业仍然使用yarn/mesos等框架管理离线作业，所以迁移作业到k8s上，或者是将yarn/mesos等框架接入到k8s中，形成多调度系统共存的局面，如何在系统上做资源调度协调，也是一个较为困难的问题。

# 参考文献

[1] Shehabi A, Smith S, Sartor D, Brown R, Herrlin M, Koomey J et al. United States Data Center Energy Usage Report. Lawrence Berkeley National Laboratory, Berkeley, California, 2016. 65 p.

[2] Amazon Blog. <https://glinden.blogspot.jp/2006/11/marissa-mayer-at-web-20.html>

[3] Verma A, Pedrosa L, Korupolu M, et al. Large-scale cluster management at Google with Borg. In: Proc. of the 10th European Conf.

on Computer Systems. 2015. 1–17

[4] Rzadca K, Findeisen P, Swiderski J, et al. Autopilot: workload autoscaling at Google[C]//Proceedings of the Fifteenth European Conference on Computer Systems. 2020: 1-16.

[5] Tirmazi M, Barker A, Deng N, et al. Borg: the next generation[C]//Proceedings of the Fifteenth European Conference on Computer Systems. 2020: 1-14.

[6] Bartholomew D J. Time Series Analysis Forecasting and Control[J]. 1971.

[7] Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.

[8] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.

[9]Autoscaler. <https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler>

[10] Fu R, Zhang Z, Li L. Using LSTM and GRU neural network methods for traffic flow prediction[C]//2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC). IEEE, 2016: 324-328.

[11] Wang T. Predictive vertical CPU autoscaling in Kubernetes based on time-series forecasting with Holt-Winters exponential smoothing and long short-term memory[J]. 2021.

[12] Ghodsi A, Zaharia M, Hindman B, et al. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types[C]//Nsdi. 2011, 11(2011): 24-24.

[13] Chen S, Delimitrou C, Martínez J F. Parties: Qos-aware resource partitioning for multiple interactive services[C]//Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. 2019: 107-120.

[14] MacQueen J. Some methods for classification and analysis of multivariate observations[C]//Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. 1967, 1(14): 281-297.

[15] Vasile M A, Pop F, Tutueanu R I, et al. HySARC 2: Hybrid scheduling algorithm based on resource clustering in cloud environments[C]//International conference on algorithms and architectures for parallel processing. Springer, Cham, 2013: 416-425.

[16] Paterek A. Improving regularized singular value decomposition for collaborative filtering[C]//Proceedings of KDD cup and workshop. 2007, 2007: 5-8.

[17] Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters[J]. ACM SIGPLAN Notices, 2013, 48(4): 77-88.

[18] Hindman B, Konwinski A, Zaharia M, et al. Mesos: A platform for fine-grained resource sharing in the data center[C]//NSDI. 2011, 11(2011): 22-22.

[19] Vavilapalli V K, Murthy A C, Douglas C, et al. Apache hadoop yarn: Yet another resource negotiator[C]//Proceedings of the 4th annual Symposium on Cloud Computing. 2013: 1-16.

[20] Burns B, Grant B, Oppenheimer D, et al. Borg, omega, and kubernetes[J]. Communications of the ACM, 2016, 59(5): 50-57.

[21] Romero F, Delimitrou C. Mage: Online and interference-aware scheduling for multi-scale heterogeneous systems[C]//Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. 2018: 1-13.

[22] Tang X, Wang H, Ma X, et al. Spread-n-share: improving application performance and cluster throughput with resource-aware job placement[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019: 1-15.

[23] Li Q, Li Y, Tu BB, et al. QoS guarenteed dynamic resource in internet data centers. Chinese Journal of Computers, 2014,37-49.

[24] Garefalakis P, Karanasos K, Pietzuch P, et al. Medea: scheduling of long running applications in shared production clusters[C]//Proceedings of the Thirteenth EuroSys Conference. 2018: 1-13.

[25] 自动化弹性伸缩如何支持百万级核心错峰混部. <https://www.infoq.cn/article/uamkSif3MYObyZulVESc>

[26] Zhang Z, Li C, Tao Y, et al. Fuxi: a fault-tolerant resource management and job scheduling system at internet scale[C]//Proceedings of the VLDB Endowment. VLDB Endowment Inc., 2014, 7(13): 1393-1404.

[27] Apache YuniKorn.https://yunikorn.apache.org/

[28]Zhang, Xiao, E. Tune, R. Hagmann, Rohit Jnagal, Vrigo Gokhale and John Wilkes. “CPI2: CPU performance isolation for shared compute clusters.” EuroSys '13 (2013).

[29] J. Mars, L. Tang, R. Hundt, K. Skadron and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2011, pp. 248-259.

[30]Baidu Inc.百度大规模战略性混部系统演进[EB/OL].[2019-08-05]. [https://www.infoq.cn/article/aeut\*zaiffp0q4mskdsg](https://www.infoq.cn/article/aeut*zaiffp0q4mskdsg).

[31]Tencent Inc. Caelus -腾讯基于 K8s 的全场景在线离线混部解决方案[EB/OL].[2021-06-15]. https://cloud.tencent.com/developer/salon/live-1459.

[32]Alibaba Inc. 想了解阿里巴巴的云化架构看这篇就够了[EB/OL].[2017-12-26].https://www.infoq.cn/article/2017/12/cloud-sigma-pouch-alibaba.

[33] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. 2013. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers. <i>SIGARCH Comput. Archit. News</i> 41, 3 (June 2013), 607–618. DOI:https://doi.org/10.1145/2508148.2485974.

[34] 王 康 瑾 , 贾 统 , 李 影 . 在 离 线 混 部 作 业 调 度 与 资 源 管 理 技 术 研 究 综 述 .软 件 学 报 ,2020,31(10).