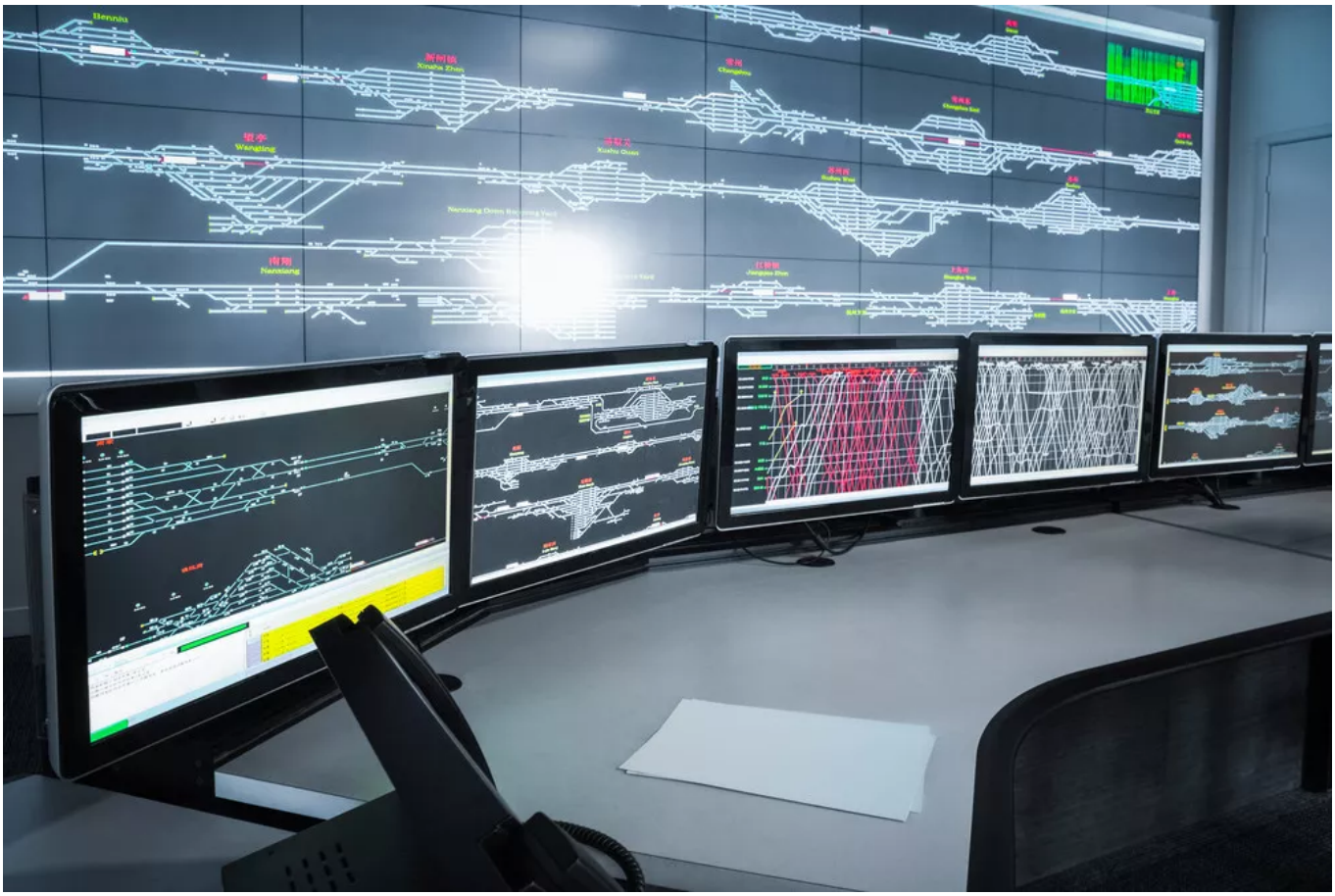
首次公开！菜鸟弹性调度系统的架构设计

原创：长陵 阿里技术 2018-03-08



阿里妹导读：菜鸟方舟(ark)是面向菜鸟所有研发的资源管理和运维平台，负责对菜鸟的基础设施资源进行管控，以支撑日常和大促的资源需求。弹性调度是菜鸟方舟的一个重要组成部分，也是方舟的一个重要的功能特性。

通过弹性调度，能够使应用在业务压力上升时及时扩充资源，而在业务压力下降时对资源进行释放，从而实现在保证稳定性的前提下尽可能地提升资源使用效率。在未来引入离线任务进行混部，或者细粒度资源计价方式后，这种模式将会大幅度降低菜鸟整体IT成本。今天，我们来细细聊聊菜鸟方舟弹性调度系统背后的技术，希望对你有所启发。

为什么菜鸟需要弹性调度？

在弹性调度出现之前，菜鸟整体资源使用率都处于一个比较低的水平，这是因为：

1.在线应用一般是通过单机性能压测，并且结合经验预估业务流量的方式来确定所需容器数量。这种方式很大程度上会受到评估者主观因素的干扰，在估算业务流量时也通常会保留较大的冗余。

2.以往的模式下，一个应用分组的扩缩容操作频率很低，这使估算业务流量时，需要以每天的峰值流量以及未来一段时间（通常以月为单位）内业务的发展情况来作为评估标准。而一般峰值流量出现时段可能只占全天时间的一小部分，非峰值时段就出现大量的资源浪费。

从接入的弹性应用分组表现来看，容量评估不准确是非常普遍的现象，而且与实际偏差值非常大。弹性调度作为一种在线动态评估系统运行状态并且做出扩缩容决策的系统，它让应用的开发者以及运维人员对资源的关注点，从具象化的容器数转换成抽象程度更高的“目标”（例如合理的资源使用率目标，服务的合理rt目标），降低了人工评估时不可避免引入的主观因素影响。另外，结合方舟平台提供的可靠、高效的扩缩容能力，对应用分组的扩缩容操作时效性可以达到分钟级，从而真正意义上实现资源的“按需使用”。对于菜鸟来说，弹性调度是提升资源使用率最为行之有效的一种方式。

为什么菜鸟更适合落地弹性调度？

弹性调度虽然能够带来较大的使用收益，但并不是适用于所有的公司或组织，而其之所以能够成功在菜鸟进行落地，主要取决于以下几点原因：

- 菜鸟的业务特点决定系统是协调商家，cp，消费者之间的信息流转，而且物流订单流转的长链路多交互的特点也决定了信息流大于实操流即可，所以我们的系统面临导购秒杀型的脉冲峰值菜鸟方舟弹性调度系统场景微乎其微。
- 菜鸟在2017年年初全面实现容器化并且接入混合云架构体系后，已经完成了资源管理从“面向机器”到“面向应用”的转变，应用的部署、扩缩容等核心运维流程得到了极大的简化和提效。方舟平台作为容器资源管控平台，在经历了2017年618、双十一等大促活动的考验后，其稳定性已经得到了充分的验证，这就为弹性调度的落地创造了充分的技术基础。
- 菜鸟的核心应用绝大多数都属于无状态的在线计算应用，每天业务压力峰谷值差距明显，这就为弹性调度的落地提供了足够的业务场景基础。
- 弹性调度并不是一项独立的工程，需要很多基础服务进行协助，并且依赖于统一、规范的系统环境。而菜鸟的应用遵从阿里集团规范，弹性调度可以直接读取alimonitor、鹰眼、alimetrics等工具提供的监控运维数据，并且核心应用所使用的技术栈基本上得到了收敛，这就位弹性调度的落地提供了充分的环境基础。

基于如上四点，菜鸟才能以较小的成本快速实现了弹性调度。

与同类产品有什么区别？

在集团内部已经有不少团队开发了针对某个业务域的弹性调度产品，而业内部分公有云服务商也提供了弹性伸缩服务，菜鸟弹性调度所面对的问题以及对应的产品思路上与这些同类产品有什么区别呢？

首先，菜鸟弹性调度所期望覆盖的应用范围是菜鸟所有的无状态核心应用，这些核心应用所涉及的业务链路、逻辑特性、资源倾向性、业务流量特性等都存在非常大的差异性，很难抽象出一种通用的业务模式来描述这些应用。因此，不同于针对某个特定的业务域的弹性调度，菜鸟弹性调度在进行设计

时不能进行过多的业务假设，在设计调度算法和策略模式时必须考虑到足够的通用性；在配置上需要给予使用者充分的个性化能力以应对不同的业务场景；在系统结构设计时，需要考虑到策略横向扩展能力，当有新的特殊业务场景出现时，能够进行快速线性扩展。

其次，在应对复杂场景时，系统越是通用，所带来的配置选项也就越多，公有云上的弹性调度服务通常提供非常多的配置参数，正是因为他们希望通过这种“把问题抛给用户”的方式，来抵消问题的复杂性，让使用者自己为自身的稳定性和成本负责。这种弹性调度带来的稳定性风险和成本节约效果完全依赖于用户本身对于这项技术的了解。与之不同，我们作为菜鸟的基础技术团队，我们把自己的角色定义为稳定性和成本问题的解决者，而不是向我们的用户抛去更多的问题，我们希望提供给菜鸟所有应用owner的不仅仅只是像公有云上弹性这样的一种新的资源管理功能，而是替我们的用户解决降低成本、提升稳定性。因此，在产品设计之初时，我们就希望绝大多数应用分组能够做到一键接入弹性，把绝大多数应用的配置问题在使用者感知之前就进行解决，将策略参数配置纳入到我们的核心职责范围内。而对于那些具有特殊性要求的应用，为其提供辅助性建议，帮助其进行少量的配置即完成弹性能力的引入。

弹性调度应用现状

截止到目前为止，菜鸟已经基本实现了对容器数量15台以上（接入前）的无状态应用分组进行弹性接入，接入应用分组的整体全天CPU平均使用率达到20%以上（计算方法为取分组CPU使用率与分组容器数的加权平均值）。每天扩缩容总容器数在3000台以上。在2017年双十一时，弹性调度作为辅助手段从11月12日0点起对部分应用分组进行缩容，使菜鸟占用物理CPU核数与包裹数的比例得到显著下降。

以下图一展示的是一个应用分组一天当中CPU使用率与容器数的变化曲线对比；图二展示的是该应用分组某个核心服务同时段流量变化曲线：

菜鸟方舟弹性调度方案介绍

弹性调度的基本模式

如前文所言，方舟的弹性调度希望提供给用户的不只是一种弹性操作集群资源的能力，而是要对所有用户的成本和稳定性优化这件事负责。由于目标应用在各方面差异性很大，所涉及的配置项数以千计并且一直处于动态变化状态，全靠我们人工进行配置管理非常不现实。

由此，方舟弹性调度提出了一种闭环反馈式的模式（如上图所示）。弹性调度基础能力基于应用分组运行情况和不同应用分组的策略配置参数，做出扩缩容决策，并通过方舟的容器操作服务调整集群容器数量；应用分组集群受到集群容器数量变化的影响，会产生不同的表现行为（例如扩容时集群平均CPU使用率会发生变化，服务rt会在一定范围内下降等）；应用分组的表现在以实时数据提供给弹性决策的同时，也会进行历史数据的离线存储（Alimonitor/EagleEye等集团标准监控系统都提供了这样的数据服务）；自动策略配置会周期性获取这些历史数据，并依照一定的算法，对不同的应用分组进行不同的策略配置，从而再次影响到弹性调度策略的决策。

这种模式的优越性在于：

1.具备一定程度的自我进化能力。当应用分组刚刚接入弹性时，其大多数的策略参数都为默认值；而当弹性运行一段时间后，结合自动评估方式，各种参数会得到不断的修正以达到更好的弹性效果。以服务安全策略为例：服务安全策略在实时决策阶段概括起来就是对当前服务rt于服务的sla阈值进行比较，刚刚接入弹性时，服务的sla是基于服务接入弹性前的历史rt来得到的，一般来说非弹性状态下服务rt的表现，与弹性状态下服务rt的表现是有很大的区别的，可能一开始由于服务sla设置得不合理（一般来说是过小），会出现“多扩”的现象，由服务rt违反sla引起的扩容会占到整体扩容原因的大多数。这种现象会被每天定时执行的分析任务捕捉到，判断出sla设置得不合理，结合最近几天的运行状态，重新计算服务sla，由此提高阈值设置的合理性；

2.以更高的抽象层次来进行海量参数的配置，以解决普遍问题。还是以服务rt的sla阈值为例，当我们把配置视角关注到一个具体服务时，我们可能会纠结于一个服务它所对应的具体业务逻辑是什么、它涉及的调用链路是什么、上游服务对它的容忍性等等细节问题，那么这样一来，面对菜鸟不同应用提供的成千上万个服务，逐一配置根本不可能做到（注：每天都会服务会上线和下线，服务的业务逻辑也可能发生变化，配置是需要进行经常性更新的，这无疑使人工配置更加变得不现实）。而自动策略配置逻辑以更高的抽象层次来看待各项参数，对于服务rt，基于一个普遍适用的假设：“服务rt在一天当中的绝大多数时间都是处于合理状态”，并且通过概率分布计算（服务rt真正的分布情况也可以通过历史数据统计得到），可以得到一个数学意义上的sla阈值（以正态分布为例，求得一段时间内服务的平均rt和rt分布标准差，即能得到在不同概率下应该设置的阈值）。

如上图的正态分布曲线，我们如果把阈值定为平均rt + 2个标准差，那么依照概率粗略计算，我们假设一天当中有将近33分钟服务处于rt过高状态（1440分钟 * （1 - 0.9544）/ 2），由此就得到了一个数学上合理的阈值（这部分逻辑只是服务安全策略逻辑的一小部分，具体在后文介绍该策略时具体说明）。这样一来，对于各式各样的服务，只要能获取到它的历史监控数据，就能自动、快速地得到这个数学上的阈值。

弹性调度的架构体系

这部分就不在此做过分冗余的解读了，本文的其他部分或多或少会涉及到。

为什么采用三层决策的模式？

首先介绍一下方舟弹性调度的三层决策：

1.第一层是策略决策，策略决策层由多个不同的策略组成，并且支持快速扩展。策略之间逻辑完全隔离，每个策略计算完成后都会独立输出动作（扩容、缩容、不变）和数量。为了能够适应不同应用之间的异构，每个应用分组也可以根据实际情况启动或关闭不同的策略。

2.第二层是聚合决策，聚合决策收集第一层所有策略的决策结果，并依据聚合规则得到一个合并后的<动作，数量>组。这一层的规则十分简单：当同时存在扩容和缩容决策结果时，以扩容为准，忽视缩容结果；当存在多个扩容结果时，以扩容数量最多的结果作为最终结果；当存在多个缩容结果时，以缩容数量少的结果作为最终结果。

3.第三层是执行决策，这部分决策主要会考虑到一些规则，最终告诉扩缩容服务：要不要扩缩，要扩缩多少个容器，如果是缩容那么要缩容哪几个具体容器，如果是扩容那么具体的容器规格、扩容到的机房等。执行决策进行判断时需要考虑到的规则非常复杂，这里简单罗列一些相对重要的规则：

- 机房均摊规则；
- 当前应用分组的扩缩容状态规则，例如如果本次为扩容：如果正在扩容，当本次扩容目标数量大于正在扩容的目标数量时，取差值再次发起一个扩容，由此实现并行扩容；当本次扩容目标数量小于正在扩容的目标数量时，忽略本次的扩容请求；若正在进行缩容，则立即停止缩容，并根据目标容器数和当前容器数发起扩容。
- 模式规则：弹性调度目前支持全自动扩缩模式、人工审批模式两种，如果当前分组为人工审批模式，那么本次决策会需要管理员进行审批。
- 最大值最小值保护规则：应用分组可以配置最大值最小值，执行决策会保证由弹性调度发起的扩缩任务，不会使最终容器数超过最大值或小于最小值。

此外，执行决策层对于单个分组来说是强一致的，并且第二层输出的决策结果，是集群需要达到的目标容器数量，这种设计是前两层能够做到完全无状态且幂等的重要因素。

三层决策器使每一层只需要关注自己本身的决策逻辑，分离了“变与不变”的业务逻辑，对扩缩容的最终确定进行层层验证，是实现“覆盖菜鸟大多数应用”目标的基础。

如何做到计算的无状态、幂等和高可用？

1.方舟弹性调度深度依赖了ISS，ISS作为一款经历过大促考验，并且为菜鸟很多核心业务提供异步任务调度服务的高可用中间件，在功能、性能和稳定性上都非常可靠。方舟弹性调度对于在线数据的获取采用了“短频周期性主动拉取”的模式，通过ISS提供的周期性异步任务调用功能，为每个应用分组在接入弹性时自动注册一个独立的ISS周期任务。ISS在发起任务时，会在目标集群中随机进行选取，

并且对任务执行时的生命周期进行管理，支持任务的重试。此外，ISS的客户端也提供资源保护能力，当集群中的某个进程压力过高时会更换目标机进行重试。

2.方舟弹性调度的在线计算数据源自于内嵌式监控系统alimetrics。alimetrics是伴随web容器的一种嵌入式metrics系统，包含非常丰富的监控项。当需要获取应用分组的细粒度监控数据时，这种数据查询、读取、传输压力是被分摊到每一个目标容器的，而非一个集中式的数据中心，这种设计使得数据源不存在单点，数据源的可靠性和压力容忍能力相比于依赖一个中心式的数据服务来说，要优越很多。

3.为了过滤毛刺，所有计算都基于或大或小的滑动时间窗口。通过alimetrics获取较短时间窗口（1小时以内）数据时能拥有非常高的性能，并且对应用的干扰非常小，这样就降低了计算的重试成本。基于这一能力，弹性调度的计算任务可以在每次执行时重新获取一个时间窗口内的全部监控数据，而不需要在自身内存中维护一个滑动窗口，这是弹性调度计算无状态的基础；

4.弹性调度三层决策器中，第三层与其他两层部署在不同的集群中。由于无论应用分组状态如何，第一层和第二层都要进行短频周期性计算，而只有在需要进行扩缩容时（只占一天中很小的一部分）才会将任务发往第三层，因此将强一致性的范围限定在第三层，在保证可靠性的同时，对性能影响最少。而第二层输出到第三层的决策数量，以“目标容器数”而非“扩缩容数量”的形式给出，这样一来，即使在同一时刻对于一个应用分组有多个弹性决策任务在执行，向第三层输出多个决策结果，也不会影响最终的扩缩容行为。

决策策略

方舟弹性调度的决策策略支持快速横向扩展，目前已经包含多个决策策略，部分策略处于测试验证状态，这里对几个最为核心，同时也是最早上线运行的策略进行介绍：

资源安全策略

资源安全策略关注的是系统资源使用情况。目前，基于以往的运维经验以及菜鸟的业务特点，资源安全策略关注CPU、LOAD1和Process Running队列三个系统参数。当其中有一项以上，在近期时间窗口内的集群平均（为了消除毛刺和流量不均带来的影响）违反上限阈值时（支持个性化配置），发起扩容。当存在多项违反时，取需要扩容数量最大的一项为当前策略决策结果（数量计算方法在后面给出）。

上图是一个资源安全策略得到扩容结果时的资源使用情况。

资源优化策略

资源优化策略同样关注的是系统资源的使用情况，但是它的存在是为了在系统空闲时回收资源。同样关注上述三个系统参数，当这三项同时低于下限阈值时，发起缩容。注意，由于第二层决策器的存在，当有其他决策器要求扩容时，资源优化策略产生的缩容需求就会被抑制。

上图是一个资源优化策略得到缩容结果时的资源使用情况。

时间策略

目前的弹性决策模式是后验的，即在发生阈值违反后，才发起扩容。对于有些业务来说，存在有规律的流量突然上扬，例如一些定时计算任务。自动策略配置基于应用的历史流量变化情况，当判定流量变化为周期性变化，且变化幅度过大，后验式弹性无法及时跟上时，会为这个分组生成一个时间策略配置，即在每天的指定时间段内，将集群容器数维持在一个指定数量之上。由于第二层决策器的存在，当其他策略在这个时间段内判断需要的容器数，大于配置的指定数量，那么以较大的一项作为结果；而在这段时间内，由于时间策略会持续生成扩容结果（如果数量已经满足，那么生成扩容决策，但数量为0），其他的缩容决策结果会被持续抑制。

服务安全策略

服务安全策略是目前最为复杂的一个策略，目前，服务包含消息队列Consumer、RPC服务、HTTP服务三种。每天有至少一半以上的扩容任务是由服务安全策略发起的。

选择qps还是rt?

很多的弹性调度系统会选择服务的qps作为最重要的一个考虑因素，但是我们在经过前期的一系列思考讨论后，决定放弃qps而使用rt，主要出于以下几点原因：

1.qps是一个服务的变量，它的变化受限于使用者的使用情况。你可以判断对于xx业务，在当前时刻的qps是否合理，但是对系统来说，只要能够承载，服务成功率和rt在合理范围内，qps取任何值都是合理的。换句话说，当前总qps可以作为业务健康度判断依据，但不能作为系统（狭义）健康程度的判断依据（单机最大可承受qps与之不同，注意区别）。rt和服务成功率才是一个服务最为根本的表现特性。

2.通过当前qps和单机最大可承受qps来得到当前容器数，在资源完全隔离，且每个query使用的资源近乎相等时才成立。而对于菜鸟的应用来说：

- 很多核心应用是跨业务链路的，服务千姿百态。一个数据查询服务和一个涉及业务操作的服务很可能出现在同一个应用分组上，此时，总qps这个概念变得毫无意义，而获取不同服务单个请求与资源的关系根本不可能。
- 目前的容器隔离效果并不是特别完美，在全链路压测时经常出现同物理机容器使用同构资源互相争抢的现象，这就使线上实际运行环境与单机性能压测的环境无法做到相同，单机性能压测数据的参考意义值得怀疑。
- 服务可能随时都在变化，而单机性能压测并不能在每次发生变化时及时跟进，时效性无法保证。

阈值比较与多服务投票

如何为海量服务的rt自动配置阈值已经在前面的例子中已经对此做过介绍，此处不再进行赘述。但是需要注意的是：由于这种阈值设定只是数学上的“合理阈值”，还需要其他手段进行修饰。因此我们又引入一个假设：如果是资源不足引起的rt违反阈值，那么该分组中所有的服务都会受到影响。这是因为应用分组内服务之间所使用的资源是没有隔离的，他们使用同一个系统环境（受到影响并不意味着所有服务都会违反阈值，不同的服务所依赖的资源种类并不相同，对资源短缺的耐受能力也不相同）。

因此，我们采用了一种“多服务投票”模式来进行rt判断。每个服务分别进行独立的阈值比较，当rt违反阈值的服务占总服务的比例达到一定程度时，才做出扩容决策，服务安全决策的扩容数量由阈值违反程度（按违反百分比来计算）最高的服务决定。

从实际运行的情况来看，当采用“只要有服务违反阈值就进行扩容”的方式运行时，出现误扩、多扩的频率非常高，而引入这种“多服务投票”模式后，误扩、多扩基本上被消除了（实际上人工判断一个扩容是否为误扩、多扩时也常常采用这种模式，发现多个服务的rt同时飙高时，则认为扩容合理）。此外，一个应用分组所提供的服务越多，这种模式运行的效果越好。

下游分析

并不是所有的rt违反阈值情况都需要扩容，如果是所依赖的中间件或下游服务导致的rt升高，扩容并不能解决问题，甚至还有可能造成更坏的影响（例如db线程池满）。因此，服务安全策略在进行计算时，如果发现一个服务违反了它的阈值，会先查询它的下游依赖服务、中间件调用rt是否违反各自的sla，只有在服务自身违反阈值，但下游服务和中间件没有违反阈值时，才会参与扩容投票。离线任务基于历史数据计算服务的sla时，同时也会计算它的依赖服务与中间件的阈值，其中链路结构数据来自于鹰眼的离线数据。

一些其他的专项问题

如何处理毛刺？

弹性调度需要保证扩容足够及时，但又要对系统的毛刺有足够的耐受能力，否则会造成误扩、误缩。毛刺在实际环境中是普遍现象，不仅流量不均、网络抖动等环境问题会导致毛刺的发生，监控统计系统潜在的异常和bug也有可能会诱发毛刺。

为此，方舟弹性调度在进行任何计算时，都会引入时间窗口数据作为计算源头，而每块时间的数据内，包含的是一个集群所有容器的数据：

计算时，对于每个所需的监控项，会去掉时间窗口内的最大值和最小值，然后求取平均，以此来抹去毛刺带来的影响。另外，时间窗口的大小对于去毛刺以及及时性也会有很大的影响。目前，方舟弹性调度中存在5分钟和10分钟两种不同规格的时间窗口。对于可能会导致扩容的策略来说，选取较小的时间窗口，而对于可能会导致缩容的策略来说，选取较大的时间窗口（我们希望扩容相对于缩容来说，更加激进）。

如何计算扩容和缩容数量？

在确定了是否要扩缩后，第二步便是确定扩缩容数量。首先介绍缩容，由于缩容速度相对于扩容较快，且缩容慢不会影响稳定性，所以方舟弹性调度的缩容采用一种“小步快跑”的方式，只要决定缩容，那么固定缩容的容器数量为集群当前容器数量的10%。

扩容数量的判断相对于缩容来说要复杂得多，我们这里以最为常见的阈值比较类策略的数量决策为例。首先，定义一个大的原则：每次扩容数量最大不得超过集群当前容器数的50%，在这个范围内，对阈值违反的程度越高，扩容的容器数量越多。因此我们在这里引入sigmoid函数，通过一定的转换使其计算结果在0~0.5之间，将阈值违反程度百分比作为入参代入sigmoid函数（当然，会进行一些参数的调整），从而得到扩容的百分比。

sigmoid函数是一个在0~1之间变化的曲线，常用于数据的归一化计算。对于f(x)=sigmoid(x) - 0.5来说，当x > 0时，该函数的取值在0 ~ 0.5之间。

对于像CPU使用率这种数据来说，有一个区别是该项的取值最大不会超过100，存在一个上限；而诸如服务rt这类的数据则不存在上限。那么显然，完全用同一个公式会导致CPU触发的扩容数量小于服务rt触发的扩容数量。此时，只需要设置一个目标，例如我们希望当CPU使用率达到80%，即违反程度为 （80% - 阈值）/ 阈值时，扩容比例接近50%，基于这个目标对f(x)进行逆运算就能得到系数a，使进行针对CPU的扩容数量计算时代入这个系数a，就能得到所期望的结果。

如何实现链路容量的弹性？

只要链路上所有应用分组都接入了弹性，那么已经可以认为此链路的容量也具备弹性。

如何应对大促的场景？

方舟通过“容器计划”和弹性调度进行配合的方式，为大促提供了完整的资源管理解决方案。容器计划可以让各个应用Owner在指定的大促时间段内（包括压测和大促时段）提出容量申请。当接入弹性的应用分组在进入大促时间段内，弹性调度的扩缩容模式会立即转变成人工确认模式，并且将会把集群容器数扩容到容器计划所申请的数量。当大促时间段结束时，非弹性的应用分组将会直接缩容回原有容器数量；而弹性应用分组则会通过弹性策略进行缓慢资源回收。

在大促时间段内，尽管弹性调度没有自动进行扩缩容，但是扩缩容决策依然在进行中。弹性调度会持续向管理员或大促决策小组发出容器扩缩容建议，一方面在流量超出预期、资源不足时及时发出警告并且提供建议的扩容数量，也能协助管理员对一些资源进行及时的回收，帮助进行缩容决策。

弹性调度如何推动业务本身的演进？

弹性调度推动业务本身的演进是我们一直以来的目标，对我们来说，只有这样才可以形成真正意义上的业务闭环。这种推动主要是采用数据分析的方式进行，目前弹性调度会产出以下几种数据：

- rt|资源相关性，反映了服务rt和CPU等系统资源使用率的相关程度，相关程度越高，则服务使用情况对资源越敏感，使用弹性调度的收益也越明显；
- 中间件稳定性，反映了由中间件rt超出阈值引起的服务rt违反。由于服务rt的升高直接影响到了服务的可用性，因此当这项数据过高时，会建议检查中间件的使用情况，进行优化；
- 下游稳定性，反映了由下游服务rt超出阈值引起的服务rt违反。当这项数据过高时，会建议当前应用推动下游服务稳定性升级，或者推动下游弹性化；
- 应用启动时效，一个扩容任务直到应用实例启动完成，能够对外正常服务才认为成功，应用启动时间越短，那么弹性调度对其扩容也就越及时，当出现流量洪峰时也能最快速度进行保护；
- 限流配置合理性。我们发现不少接入的应用在服务rt都处于正常范围，且系统资源使用率极低的情况下居然触发了自身所配置的限流值，对于这种情况，会建议应用进行合理的业务分析和测试来调整限流值配置。

未来发展

目前，方舟的弹性调度还处于一个发展成长的过程中，对于一些应用的调度效果还要进行进一步的提升。我们也期待更多了解容器、调度、中间件技术的的童鞋加入，一起并肩作战。简历投递邮箱：jian.weng@alibaba-inc.com 我们翘首以盼～

可爱的阿里实习生，你在哪?

（附申请指南+官方攻略）

揭秘！双11万亿流量下的分布式缓存系统 Tair

敏捷开发的根本矛盾是什么?

从业十余年的工程师在思考

关注「阿里技术」

把握前沿技术脉搏