# AHPA：阿里云容器服务 Kubernetes 版上的自适应水平 Pod 自动扩缩容系统

**Zhiqiang Zhou**[1], **Chaoli Zhang**[1], **Lingna Ma**[1],**JingGu**[3],**Huajie Qian**[2],**Qingsong Wen**[2], **Liang Sun**[2],**PengLi**[3],**Zhimin Tang**[3]阿里巴巴集团达摩院，中国杭州[2]阿里巴巴集团达摩院，美国华盛顿州贝尔维尤 [3]阿里巴巴集团阿里云，中国杭州{zhouzhiqiang.zzq, chaoli.zcl, malingna.mln,zibai.gj, h.qian, qingsong.wen, liang.sun, yuanyi.lp, zhimin.tangzm}@alibaba-inc.com

arXiv:2303.03640v1 [cs.LG] 7 Mar 2023

**摘要**

Kubernetes 中现有的应用程序实例资源分配策略无法根据业务需求动态调整，导致业务波动期间资源大量浪费。此外，新型云服务的出现也对资源管理提出了更高要求。本文探讨了阿里云容器服务中的水平 Pod 资源管理，并介绍了一种新部署的人工智能算法框架 AHPA——自适应水平 Pod 自动扩缩容系统。AHPA 基于稳健分解预测算法和性能训练模型，可提供最优的 Pod 数量调整方案，从而减少 Pod 资源并维持业务稳定。自 2021 年4月部署以来，该系统已拓展至多个客户场景，包括物流、社交网络、人工智能音视频、电子商务等。与以往算法相比，AHPA 解决了弹性滞后问题，使 CPU 使用率提高了 10%，资源成本降低了 20% 以上。此外，AHPA 可根据预测的业务量自动进行弹性规划，无需人工干预，显著节省运维成本。

## 引言

云计算技术的持续发展，为当前的计算机在线服务带来了更多可能性，用户对云弹性也有了更高的期望。此外，随着云原生技术中容器服务的发展（Brewer 2015；Balalaie、Heydarnoori 和 Jamshidi 2016），越来越多基于容器服务的新型应用程序应运而生。与虚拟机（Altintas 等，2005）时代的应用程序不同，当时分钟级的人工操作已足够，而新兴应用程序通常要求秒级操作。同时，许多快速发展的应用程序具有显著的周期性波动（Yan 等，2021；Higginson 等，2020；Cortez 等，2017；Atikoglu 等，2012；Chen 等，2008），例如互联网直播、在线学习、网络游戏等。这类应用程序的业务需求反复出现高峰与低谷，因此需要以低延迟灵活地利用资源，

版权所有 © 2023，人工智能促进协会（www.aaai.org）。保留所有权利。

另一个主要的新型应用类别是无服务器计算（Baldini 等，2017；McGrath 和 Brenner，2017），它使客户端能够构建和运行服务，而无需考虑服务器。面对冷启动、低延迟和扩缩容效率等难题，要为这类应用程序弹性地管理资源颇具挑战。因此，主动自动伸缩广受欢迎（Taft 等，2018；Rebjock 等，2020；Rzadca 等，2020；Tsoumakos 等，2013）。

云技术成熟度的提升以及新兴业务的出现，都推动了高效资源利用的发展。具体而言，新的应用程序特性要求对自动伸缩进行专门设计，从而能够调整应用实例，在云原生理念下提高资源利用率。Kubernetes（k8s）（Burns 等人，2022年）是由云原生计算基金会维护的主要开源容器编排系统，用于应用程序/软件的部署和管理。可扩展性是 Kubernetes（k8s）的核心要求之一。自动伸缩（Lorido-Botran、Miguel-Alonso 和 Lozano，2014年；Qu、Calheiros 和 Buyya，2018年）是 Kubernetes 平台确保可扩展性的必要功能。通过适当配置，自动伸缩可以节省时间、防止性能瓶颈并避免资源浪费。

目前，在 Kubernetes 中有三种较为保守的应用实例数量管理方式，其流程如图 1 所示：固定实例数量、HPA（Nguyen 等，2020）和 CronHPA[1]。固定实例数量的方式在需求波动时不会改变实例数量，而 HPA 和 CronHPA 则可以根据需求变化调整实例数量。然而，这些方式在效率、简便性和准确性方面都存在不足，也都无法弹性地应对需求波动。具体而言，固定实例数量的策略最容易实施，因此得到了广泛应用。但在业务需求低谷期，这种方式也有明显缺点：会造成资源浪费。与静态方式相比，HPA 会在需求变化后调整实例数量。因此，它对需求波动的响应存在滞后，可能导致处理无效、服务质量低下，

[1]https://www.alibabacloud.com/help/en/container-service-for-kubernetes/latest/cronhpa-186978

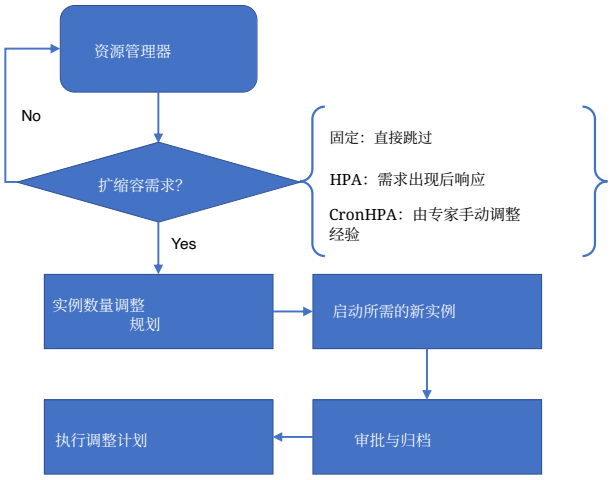



图1：Kubernetes 中资源管理的流程，包含三种保守方式（固定、HPA 和 CronHPA）。

甚至更糟，导致应用程序终止。此外，CronHPA 策略要求具备专家经验的人员手动设置扩缩容计划，这可能导致计划不准确、缺乏灵活性，并大量占用人力资源。

为了解决上述现有方案的不足，我们设计并部署了一套名为自适应水平 Pod 自动伸缩系统（AHPA）的新系统，该系统应用于阿里云容器服务 Kubernetes 版（ACK[2]），支持更出色的预测性自动伸缩。AHPA 通过分解方法准确预测下一阶段的业务情况，解决了现有方法无法动态调整 Pod 资源的问题。同时，AHPA 能够学习业务负载与所需 Pod 数量之间的映射关系，并提供最终的扩缩容行动计划；而 CronHPA 则需要人工干预。自动化模型使 AHPA 易于部署和扩展，并显著节省运维成本。

自2021年4月以来，AHPA 系统已在整个阿里云服务中部署。与云服务之前使用的算法相比，它显著提升了弹性资源管理能力。具体而言，在智能语音场景中，AHPA 当前每天的调用量超过 5000 次。它每天可节省约一万元的 Pod 资源成本，与原方案相比，直接带来超过 28% 的日均成本节省。

## 应用描述

本节将介绍 AHPA 系统的架构，如图 2 所示。该系统由两大部分组成：弹性指标和弹性对象。

弹性指标提供了多种指标，以满足客户端的不同需求，包括资源指标（例如 CPU 和内存）、自定义指标（例如，

[2]https://www.alibabacloud.com/product/kubernetes

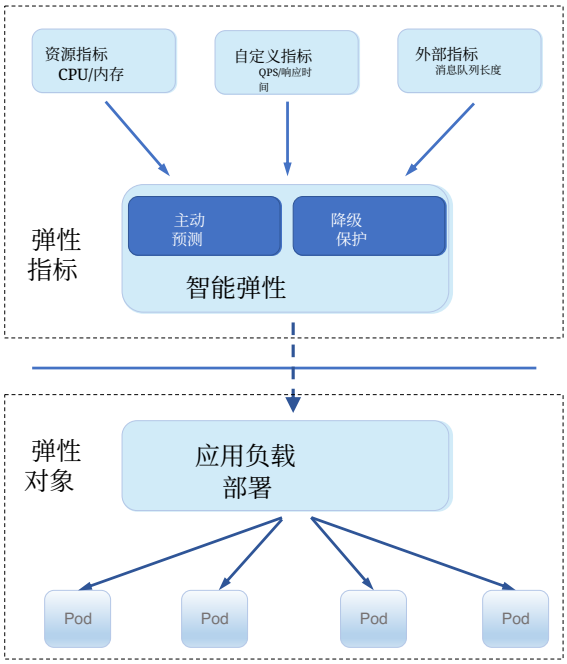



图 2：AHPA 系统的架构。

每秒查询率和响应时间）以及外部指标（例如消息队列长度）。主动预测模块用于预测未来需求，降级保护模块则旨在确保稳定性，因此智能弹性模块会输出资源预测结果。根据预测结果，弹性对象模块将通过应用部署分配所需数量的 Pod。

弹性资源管理的主要目的是，在满足服务稳定性要求的同时为客户节省成本，并降低运维中的人力资源成本。针对上述需求，我们设计了新的算法 AHPA，它具有三个典型特征，能够充分满足弹性需求和稳定性要求：

- 稳定性：AHPA 在确保客户端服务稳定性的条件下执行。
- 运维零成本：无需额外的运维工作，不会增加更多控制器，配置也比 HPA 更简洁。
- 无服务器应用可行性：AHPA 在 Pod 级别提供资源，无需考虑节点级别的使用率，从而延长应用程序的运行时间。

## 人工智能技术的应用

如图 3 所示，人工智能技术在自适应水平 Pod 自动扩缩框架中的应用主要集中于以下三个模块：未来工作负载预测、性能模型训练和扩缩容计划生成。

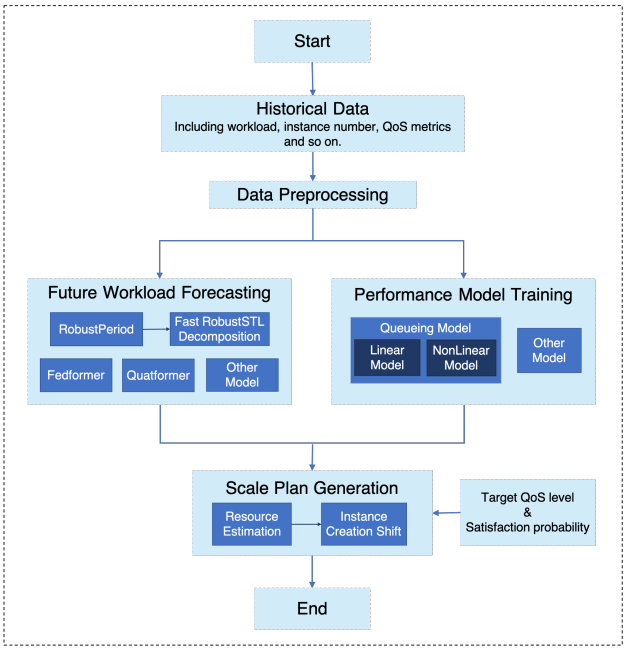



图 3：AHPA 中扩缩容算法的框架。

下面将详细介绍各个组件。

## 未来工作负载预测

第一个模块名为未来工作负载预测，严重依赖人工智能技术，在 AHPA 框架中发挥着至关重要的作用。准确预测未来工作负载可以显著提高水平 Pod 计划的调度最优性，从而通过节省系统成本获取更多收益。然而，阿里云服务中的预测过程仍面临若干复杂挑战：

• 缺失值和噪声数据：造成这一问题的原因有很多。例如，当云服务分布式系统中的某些节点受损，或用户与服务系统交互过程中发生意外时，都可能导致有效数据缺失。此外，如果采集的数据距离当前时间太久，也可能影响预测准确性。因此，设置阈值以判断数据有效性是必不可少的步骤。在某些情况下，还需要采用适当的方法补全缺失数据并对数据尺度进行归一化。

• 数据规模有限：Kubernetes 中的指标数据通常使用 Prometheus 存储。在兼顾成本和效率的情况下，一般业务数据的存储周期为 7 天。7 天的数据量作为训练集过小，训练出的机器学习/深度学习模型通常准确率较低。如何利用有限的历史数据有效估计未来的业务量，值得进一步探讨。

• 高复杂性：一般而言，用户需求变化频繁，这使数据的复杂性显著增加。例如，数据可能具有多周期等复杂特征。

因此，复杂数据对算法模型进行准确预测的能力提出了更高要求。

针对上述挑战并满足较高的预测延迟要求，我们设计了一种基于稳健分解的统计方法作为主要预测方案。具体而言，我们采用此前发表的三种稳健时间序列分解算法（Wen 等，2021、2019b、2019a）作为预处理步骤，如图 4 所示。首先，基于最大重叠离散小波变换（MODWT）解耦多个周期性的 RobustPeriod（Wen 等，2021）用于检测输入时间序列是否具有周期性及其周期长度。根据周期检测结果，针对不同的数据特征有两种处理方式。对于周期性数据，采用 RobustSTL（Wen 等，2019b、2020）将输入时间序列分解为趋势、季节性（周期分量）和残差项。对于非周期性数据，采用 RobustTrend（Wen 等，2019a）将输入时间序列分解为趋势和残差项。

从数学角度来看，上述稳健预测模块将时间序列数据分解为趋势项 $\tau_t$、周期项 $s_{i,\ t}$（如果具有周期性）和残差项 $r_t$，其公式如下：

$$y_t = \tau_t + \sum_{i=1}^{m} s_{i,t} + r_t, t = 0, 1, ..., N-1 \qquad (1)$$

历史周期项 $s_{i,\ t}$ 直接向右平移，作为未来周期项的预测值。趋势项$\tau_t$ 通过指数平滑等经典时间序列模型进行预测，以获得未来趋势分量的预测值。最后，残差部分使用分位数回归森林来获得未来残差的上界预测。将上述三个部分结合起来，即可得到下一时刻的最终预测值 $y_{t+1}$：

$$y_{t+1} = \sum_{i=1}^{m} s_{i,t} + \text{ExponentialSmoothing}(\tau_t) + \text{QuantileRegression}(r_t) \qquad (2)$$

除了上述基于稳健分解的预测方案外，对于数据量充足的场景，我们还考虑了基于Transformer的先进深度学习预测模型（Wen等人，2022年）。具体而言，我们纳入了近期开发的FEDformer（Zhou等人，2022年）模型，该模型适用于长期时间序列预测场景；以及近期开发的Quatformer（Chen等人，2022年）模型，该模型主要针对具有复杂周期模式的数据特征而设计。接下来，我们将继续完善整个方案，并添加更多能够适应不同情况的预测方法。

## 性能模型训练

获取预测的容量值后，第二个模块，即性能模型（Performance Model），用于模拟

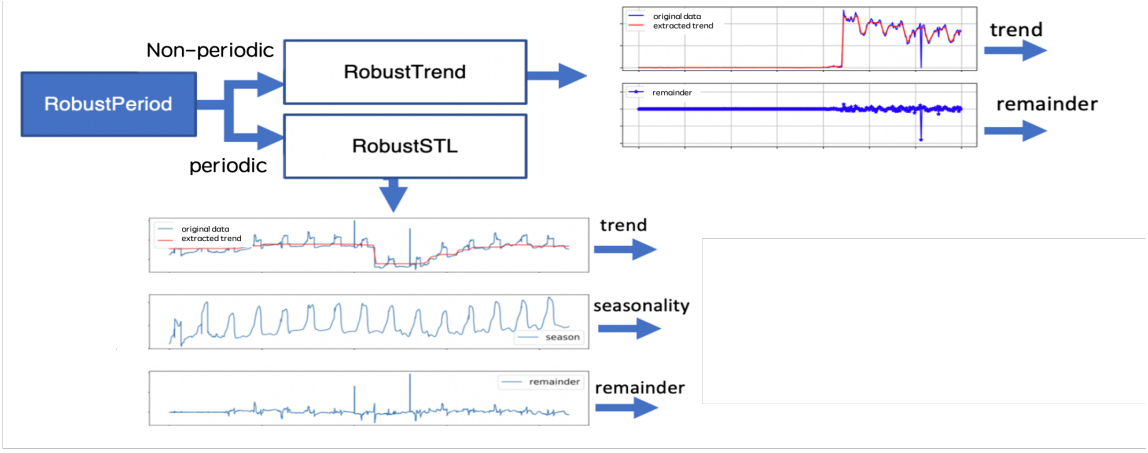



图4：AHPA 中预测模块的稳健时间序列分解。

指标度量与 Pod 数量之间的关系。在本节中，鉴于某些特定业务的实际情况，我们假设 Pod 是等效的计算单元。该模型主要采用运筹学中的排队论方法，包括两种不同的排队模型。一种是使用并行 M/M/1队列的线性关系，另一种是带有公共缓冲Pod的 M/M/c队列。

在 Pod资源利用率排队模型中，我们将业务 QPS（每秒查询数）视为系统的队列速率，并将 Pod 数量视为服务台数量 $c$（在 M/M/1队列中，每个 Pod 都可视为一个独立的服务台，且它们的处理速率相同），而 $u_i$ 是服务台的平均服务能力，可用 Pod CPU 表示。基于这些指标，我们旨在求出与每次请求的响应时间相一致的每位客户的平均等待时间。不同排队模型下的平均响应时间可通过以下数学公式表示：

$$\begin{aligned} \text{Avg}RT(M/M/1) &= (u - \frac{QPS}{N})^{-1} + \text{otherlatency} \\ \text{Avg}RT(M/M/c) &= f(QPS, u, N) + \text{otherlatency} \end{aligned} \tag{3}$$

其中，$u$ 表示单个 Pod 每秒可处理的查询数，$f$ 由 Erlang 的 C 公式（Janssen、Leeuwaarden 和 Zwart, 2011）以及 Little定律（Little 和 Graves，2008）给出。通常，我们应在满足平均响应时间（RT）要求的前提下，确定 Pod 数量的最小值。对于线性M/M/1模型，$pod = C * QPS$；对于非线性M/M/c队列模型，$pod = g(QPS)$。当可调整的 Pod 数量较少时，M/M/c 模型的表现优于另一种模型；而当可用 Pod 数量较多时，M/M/1 队列模型表现更佳。因此，在实际部署中，可以根据不同需求选择不同的模型。总之，该性能训练模块将前一个工作负载预测模块的预测值和历史指标数据作为输入，并输出需要调整的 Pod 数量。

## 扩缩容计划生成

我们的系统根据未来工作负载预测结果和性能模型生成最终的扩缩容决策。扩缩容计划包括应增加或减少的 Pod 数量及次数。

首先，我们预测满足客户端在特定指标（例如响应时间或 CPU使用率）方面需求所需的 Pod 数量。然而，Pod 启动需要一定的开销时间，这意味着系统在增加资源时会受到时间延迟问题的影响，且部署吞吐量有限。面对上述约束，我们采用改进的预测偏移算法（Flunkert 等，2020年）来应对这一挑战。

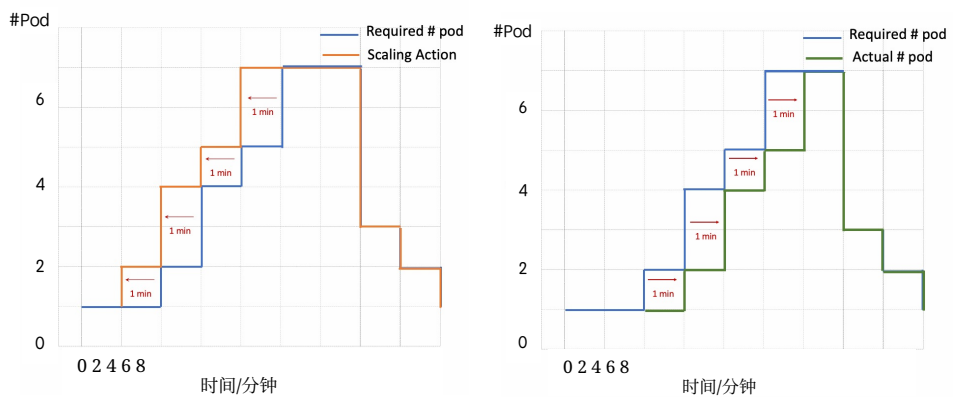
图 5 展示了一个简单示例。假设等待时间（包括 Pod 启动等）为 1 分钟；图 5(a) 表明，如果按照需求在实时精确地扩缩所需的 Pod 数量，实际可用的 Pod 数量将会有所延迟。因此，如图 5(b) 所示，应提前执行 Pod扩缩容操作。

然而，出于稳定性考虑，频繁操作并不是一个好的选择。通常，运行频率上限设置为最多每 3 分钟或每 5 分钟执行一次。因此，如何在相应时间间隔内整合调度计划，也是一个值得探讨的问题。

## 实验评估

对 AHPA 系统进行适当验证颇具挑战性，因为这些模型旨在针对不同的部署调整水平 Pod 调度。无法同时将不同模型应用于同一个应用程序。本文提供了从阿里云容器服务 Kubernetes（ACK）1.20.11 版（采用阿里云 Linux/CentOS 操作系统）的多个部署中收集的真实历史数据。时间序列数据长度为 20160，即连续 14 天每分钟一个数据点。与经典 HPA 算法和 FixPod 策略进行对比的数值结果将在以下小节中展示。

如图 6 所示，数据集**NP/WP/SP** 分别表示时间序列数据没有周期性、




(a) 实际待处理 Pod 数量　　(b) Pod 扩缩容操作

图5：扩缩容行动计划示例。

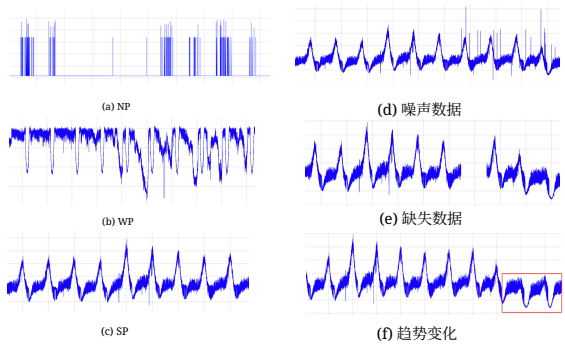



图6：从六个真实数据集中采样的典型信号序列。

弱周期性和强周期性这三种场景。对于**噪声数据** 数据集，系统底层网络链路重建偶尔会导致网络抖动，因此出现了许多异常值。如上所述，节点或监控系统发生损坏会导致缺失值，这种情况很常见；因此，我们在**缺失数据**数据集中随机选取一天并删除当天的所有数据点，以此测试该框架。最后，我们还考虑另一种常见的挑战性情况：时间序列的趋势发生变化，这种情况主要由应用程序发布新版本所致。因此，我们希望评估 AHPA 框架应用于**趋势变化**数据集时的性能。

由于云容器系统中的所有模型都旨在以最低成本满足服务质量（QoS），因此我们考虑以下三个指标来评估这些模型：成本、违规率（VR）和最大 Pod 数量。违规率（**VR**）用于评估每个模型的 QoS，即对于特定的指标目标（例如将 CPU 使用率保持在 50% 以下），VR 的计算方式为 CPU 使用率超过 50% 的时间除以总时长。**成本**是随时间提供的 Pod 总数，计算方式为 Pod 数量关于时间（分钟）的积分。根据服务级别协议（SLA）是服务提供商与客户之间的合同，其中记录了服务提供商将提供的服务，并规定了服务提供商必须达到的服务标准；在云服务的所有标准中，违规率（**VR**）发挥着最为关键的作用。

## 对比实验

在本小节中，我们考察 AHPA 框架在复杂数据的不同情况下的性能，例如具有强周期性、周周期性或无周期性的时间序列数据。

从表1所示的结果中，我们可以观察到，FixPod 策略在所有情况下都提供了零违规策略，但同时造成了 Pod 资源的大量浪费。然而，对比 HPA 和 AHPA 框架，结果表明，在成本相当的情况下，AHPA 实现了比 HPA 好得多的服务质量调度计划。当应用于具有更明显周期性特征的时间序列时，AHPA 的优势更加突出。在图7中，绿色线和橙色线分别表示 AHPA 和 HPA 方法建议的 Pod 数量，蓝色线表示给定实际工作负载和目标指标值时的最优 Pod 数量（例如，将 CPU 使用率保持在 50% 以下）。我们显然可以观察到，HPA 的调整总是存在滞后，这可能导致工作负载峰值期间的服务体验较差。此外，通过比较绿色线和橙色线，我们可以看到绿色线比橙色线平滑得多，这体现了 AHPA 算法的另一项优势：Pod 数量越平滑，在扩容和缩容 Pod 资源等操作上花费的成本就越低。

## 鲁棒性实验

在本小节中，我们评估了 AHPA 框架应对真实云服务中复杂情况的鲁棒性。这里，我们在包含噪声、异常值、缺失值和突发趋势变化的实际数据上运行该算法。

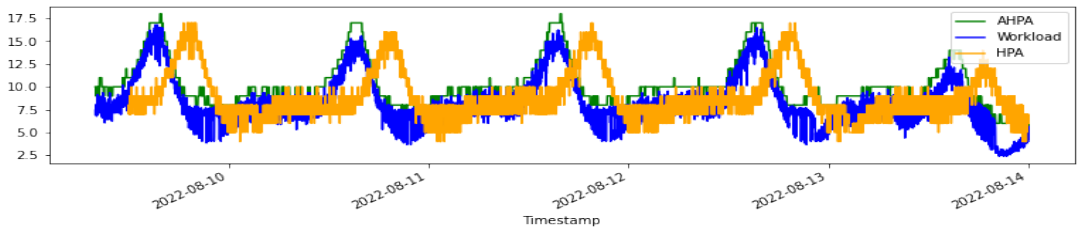



图7：Pod 数量对比。

| 数据集 | NP | | | WP | | | SP | | |
|---|---|---|---|---|---|---|---|---|---|
| 指标 | FixPod | HPA | AHPA | FixPod | HPA | AHPA | FixPod | HPA | AHPA |
| Cost | 48830 | 19657 | 22063 | 604680 | 447096 | 495541 | 201340 | 77464 | 92930 |
| VR | 0 | 0.0047 | 0.0045 | 0 | 0.3975 | 0.0695 | 0 | 0.2950 | 0.0266 |
| 最大 Pod 数 | 10 | 10 | 11 | 60 | 60 | 67 | 20 | 17 | 18 |

表 1：不同数据结构下指标/成本/违规率/最大 Pod 数的比较。

| 数据集 | 噪声数据 | | 缺失数据 | | 趋势变化 | |
|---|---|---|---|---|---|---|
| 指标 | HPA | AHPA | HPA | AHPA | HPA | AHPA |
| Cost | 77590 | 93011 | 75966 | 92519 | 77464 | 92930 |
| VR | 0.2958 | 0.0276 | 0.2934 | 0.0307 | 0.2950 | 0.0266 |
| 最大 Pod 数 | 31 | 28 | 17 | 18 | 17 | 18 |

表 2：鲁棒性测试对比。

从表 2 的结果中可以观察到，HPA 框架的违规率约为 30%，这意味着其服务质量有近三分之一的时间低于客户预期。该数值几乎比 AHPA 高出 1000%。此外，根据 AHPA 框架的结果，我们得出结论：在噪声数据、缺失值和趋势变化场景下，AHPA 框架在违规率和成本节省方面均表现出良好的鲁棒性。

## 应用使用与收益

自 2021年4月 起，AHPA 系统已部署在阿里云各项服务中，并自 2022年3月 起推广至许多不同的云服务场景。许多相关服务器已使用该系统管理弹性 Pod 资源。迄今为止，AHPA 已应用于多个客户场景，涵盖物流、社交网络、人工智能音视频、电子商务、在线教育、体育直播+和零售。大多数此类应用程序都需要应对实时高性能、低延迟以及大幅且周期性业务负载波动带来的挑战。本节将讨论 AHPA 系统的影响。

以两个应用程序为例。对于在线教育，采用 AHPA 系统的 ACK 提供以下特性：分钟级部署、行业领先的可靠性（承诺单个实例的可用性达到 99.999%）、易于管理，以及高可扩展性（根据实时需求进行扩缩容）。对于电子商务直播，我们的系统提供具备快速部署、低成本和超低延迟等特性的服务（2000+ 个节点，带宽为 150Tbps，使流媒体延迟低于 2 秒）。

客户业务应用采用 AHPA 系统部署后，弹性滞后问题得到显著缓解，CPU 使用率提升了 10%，资源成本降低了 20% 以上。目前，我们的 AHPA 算法相较于其他模型具有明显优势。此外，它能够根据业务量变化趋势自动进行灵活规划，无需人工干预，从而显著节省运维成本。使用数字数值计算运维成本对应的人力并不容易，但自动化的 AHPA 算法能够大幅解放人力，使整个系统更加高效且通用。具体而言，以智能语音场景为例，AHPA 目前每天约被调用 6,000 次，并执行约 1,000 次调度。与另一种原有的管理方式相比，AHPA 的结果显示 CPU 使用率提升了约 9%，并且每天可节省约一万元的 Pod 资源成本（件数 * 分钟数）。借助 AHPA，人工智能语音业务可节省原成本的约 28%。

## 应用开发与部署

“人工智能技术的使用”部分介绍 AHPA 系统算法部分所使用的人工智能技术的详细信息。本节将展示该算法如何开发，并部署到实际的云容器系统中。我们将首先回顾 AHPA 的主要组件（图 8），然后介绍其在整个 Kubernetes 系统中的工作细节（图 9）。

### 系统框架

如图 8 所示，AHPA 的框架包含三个主要部分：数据收集、预测和扩缩容。预测部分的主要技术和细节将在人工智能技术的应用一节中介绍。具体而言，预处理对应图 3 中的数据预处理模块；扩缩容部分包含图 3 中的工作负载预测和性能模型训练模块；修订模块则由图 3 中的扩缩容计划生成模块组成。

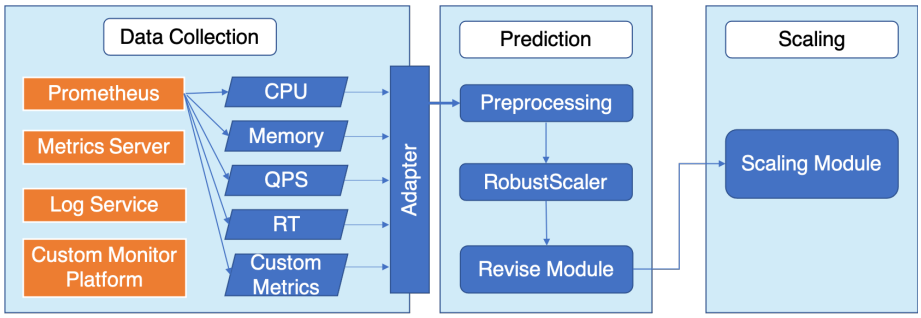



图8：AHPA 的框架。

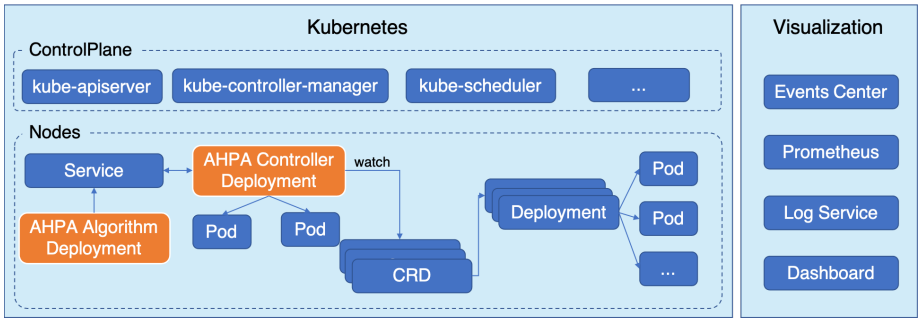



图9：AHPA 在 Kubernetes 中的部署架构。

在预测之前，数据收集模块会从各种来源收集数据，并将数据转换为统一格式。收集的数据包括资源指标（CPU、内存等）、自定义指标（QPS、响应时间等）以及其他外部指标。数据来源包括 Prometheus（一种开源的系统监控和告警工具包，可将指标信息存储为时间序列数据）、Metrics Server（K8s 的可扩展、高效容器资源指标来源）、日志服务（阿里巴巴集团开发的完整实时数据日志服务）以及其他自定义监控平台。之后，适配器模块会将不同的指标转换为统一格式。

AHPA 的最终模块是 Scaling，用于根据资源估计结果对 Pod 进行扩缩容。该模块提供两种扩缩容机制：自动模式和观察者模式。自动模式根据估计结果调整 Pod 数量；观察者模式是为客户端提供的试运行模式，用于监控 AHPA 是否按预期运行。

## 系统部署

AHPA 使用 Go 编程语言部署在阿里云容器服务 Kubernetes（ACK）上。AHPA 的部署架构如图9所示，主要包含两个组件：AHPA 算法部署和 AHPA 控制器部署。

引入自定义资源定义（CRD）来部署 Pod 扩缩容。CRD 可灵活适配不同的配置。CRD 的核心参数如下：

- 扩缩容目标引用：位置参数，适用于特殊对象的部署。



图10：AHPA 仪表盘中的可视化示例。

- 指标：位置参数，扩缩容指标，例如 CPU、内存、响应时间、QPS、GPU 等。
- averageUtilization：必填，目标阈值。例如，40 表示 CPU 利用率不能超过 40%。
- scaleStrategy：可选，扩缩容机制，可选择自动模式或观察者模式。自动模式表示启用 Pod 扩缩容，而观察者模式表示不启用 Pod 扩缩容，仅观察 AHPA 是否按预期工作。
- maxReplicas：必填，执行扩缩容时实例的最大数量。
- minReplicas：必填，执行扩缩容时实例的最小数量。
- instanceBounds：可选，扩缩容的时间范围，包括开始时间和结束时间。

• cron：可选，用于定时任务的设置。

AHPA 服务的部署也采用了独特的高可用性设计。尽管复杂系统中总会发生异常，我们的目标是提供高可用性。当 Pod 发生故障时，故障 Pod 将被终止，并创建一个新的 Pod。此外，随着应用程序或服务数量的增加，算法部署部分和控制器部署部分都可以水平扩展，以满足高并发需求。

为确保业务任务不受 AHPA 更新的影响，算法和控制器通过服务进行通信，因此可以独立更新。更新以滚动方式逐个替换 Pod，这意味着只有在新创建的 Pod 按预期正常运行后，旧 Pod 才会被终止。

我们提供了丰富的**可视化**组件，帮助客户端监控 AHPA 的状态，包括 Kubernetes 事件中心、Prothemetheus、日志服务、仪表盘等。仪表盘和日志服务的可视化示例分别如图 10 和图 11 所示。

通过仪表盘的面板界面（如图 10 所示），可以实时监控工作负载、CPU、内存等关键指标。此外，日志服务（如图 11 所示）会提供每次扩缩容操作的信息，包括扩缩容操作的时间、扩缩容操作是否处于试运行模式、实际 Pod 数量、 Pod 数量限制等。所有这些信息都有助于客户端监控 AHPA 是否按预期运行，并及时定位意外故障的原因。

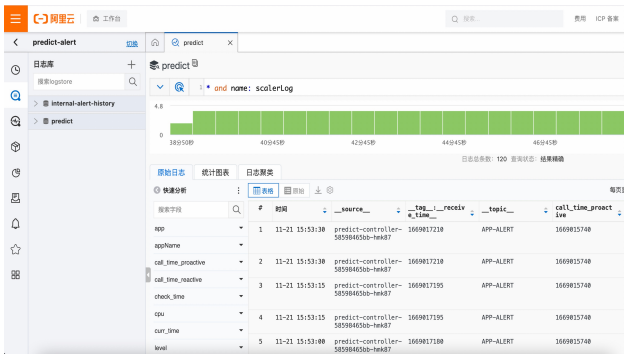


图11：日志服务中 AHPA 可视化的示例。

## 维护

随着时间推移，数据流特征的变化要求动态调整模型的超参数。不过总体而言，AHPA 无需进行大量后续运维修改。也就是说，在工作负载预测模块中，由于深度学习模型具备自学习能力，添加新数据后，无需对原始模型进行太多修改，只需根据新数据对模型进行微调或重新训练。此外，在接下来的两个模块中，最终的 Pod 数量调整方案也可以根据实际情况进行调整，而不涉及模型修改。

整个算法的核心思想是尽可能准确地预测业务，并提前应对波动以节省资源。因此，AHPA 上线后无需进行任何重大修改。在我们的实践中，截至2021年4月，只需微调部分模型参数。

## 结论与未来工作

本文介绍了改进后的框架 AHPA（自适应水平 Pod 自动伸缩），旨在为阿里云容器服务 Kubernetes 提供更好的资源管理，在节省资源的同时保障用户体验。该平台已广泛应用于阿里云服务，并部署在各种业务场景中，不局限于 ACK 容器管理。通过准确预测下一时刻的业务量，并根据历史数据获取映射关系，AHPA 能够应对节省资源与确保业务稳定之间的挑战。AHPA 算法的核心部分采用基于稳健分解的时间序列预测模块和排队论进行性能建模。自 2021年4月 部署以来，AHPA 已帮助阿里云服务中的 20 多种不同业务场景解决弹性滞后问题，并将 CPU 效率显著提升了 10%。此外，AHPA 还展现出显著优势：无需人工干预即可自动为系统提供规划。阿里云其他部门进一步推广 AHPA 的计划也已安排。

在后续工作中，为适应不同的业务场景，我们将根据具体需求，研究如何借助排队论以外的适当理论来抽象映射关系。此外，随着云服务的蓬勃发展，越来越多的大规模需求不断涌现。如何在确保各个具体业务场景效率的同时，提供适应性更强的框架，也是我们的追求目标。

## 参考文献


Altintas, Y.; Brecher, C.; Weck, M.; and Witt, S. 2005年. 虚拟机床. *CIRP* 年刊, 54(2): 115–138.

Atikoglu, B.; Xu, Y.; Frachtenberg, E.; Jiang, S.; and Paleczny, M. 2012年. 大规模键值存储的工作负载分析. 收录于 第 *12* 届 *ACM SIGMETRICS/PERFORMANCE* 计算机系统测量与建模联合国际会议论文集（*SIGMETRICS '12*）,53–64.

Balalaie, A.; Heydarnoori, A.; 和 Jamshidi, P. 2016年。微服务架构助力 DevOps：迁移到云原生架构。*IEEE Software*, 33(3)：42–52。

Baldini, I.; Castro, P.; Chang, K.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Slominski, A.; 等. 2017年. 无服务器计算：当前趋势与开放问题。载于云计算研究进展，1–20。Springer。

Brewer, E. A. 2015年. Kubernetes 与云原生之路。载于第六届 *ACM* 云计算研讨会论文集，167–167。Burns, B.; Beda, J.; Hightower, K.; and Evenson, L. 2022年.*Kubernetes*：权威指南。 ” O’Reilly Media, Inc.”。Chen, G.; He, W.; Liu, J.; Nath, S.; Rigas, L.; Xiao, L.; and Zhao, F. 2008年. 面向连接密集型互联网服务的能耗感知服务器配置与负载调度。载于第 *5* 届 *USENIX* 网络系统设计与实现研讨会论文集，NSDI ’08，337–350。美国：USENIX Association。Chen, W.; Wang, W.; Peng, B.; Wen, Q.; Zhou, T.; and Sun, L. 2022年. 学习旋转：用于复杂周期性时间序列预测的四元数 Transformer。*KDD 2022*年。Cortez, E.; Bonde, A.; Muzio, A.; Russinovich, M.; Fontoura, M.; and Bianchini, R. 2017年. 资源中心：理解并预测工作负载，以改善大型云平台中的资源管理。载于第 *26* 届操作系统原理研讨会（*SOSP ’17*）论文集，153–167。Flunkert, V.; Rebjock, Q.; Castellon, J.; Callot, L.; and Januschowski, T. 2020年. 面向大规模云应用的简单有效的预测性资源伸缩启发式方法。*arXiv* 预印本 *arXiv:2008.01215*。Higginson, A. S.; Dediu, M.; Arsene, O.; Paton, N. W.; and Embury, S. M. 2020年. 使用时间序列分析和机器学习进行数据库工作负载容量规划。载于*2020*年 *ACM SIGMOD* 数据管理国际会议论文集（*SIGMOD ’20*），769–783。Janssen, A.; Leeuwaarden, J.; and Zwart, B. 2011年. Halfin-Whitt 区域中 Erlang C 公式的下界。排队系统，68(3-4): 361–363。Little, J.; and Graves, S. C. 2008年. Little定律。*Springer US*。Lorido-Botran, T.; Miguel-Alonso, J.; and Lozano, J. A. 2014年. 云环境中弹性应用自动伸缩技术综述。网格计算杂志，12(4): 559–592。McGrath, G.; and Brenner, P. R. 2017年. 无服务器计算：设计、实现与性能。载于*2017*年 *IEEE* 第 *37* 届分布式计算系统国际会议研讨会（*ICDCSW*），405–410。IEEE。Nguyen, T.-T.; Yeom, Y.-J.; Kim, T.; Park, D.-H.; and Kim, S. 2020年. Kubernetes 中用于弹性容器编排的水平 Pod 自动伸缩。*Sensors*，20(16): 4621。Qu, C.; Calheiros, R. N.; and Buyya, R. 2018年. 云中的 Web 应用自动伸缩：分类与综述。*ACM* 计算调查（*CSUR*），51(4): 1–33。Rebjock, Q.; Flunkert, V.; Januschowski, T.; Callot, L.; and Castellon, J. 2020年. 面向大规模云应用的简单有效的预测性资源伸缩启发式方法。载于*VLDB 2021*年数据库系统与应用中的应用人工智能研讨会（*AIDB ’21*）。Rzadca, K.; Findeisen, P.; Swiderski, J.; Zych, P.; Broniek, P.; Kusmierek, J.; Nowak, P.; Strack, B.; Witusowski, P.; Hand, S.; 等. 2020年. Autopilot：工作负载自动伸缩

谷歌。在第十五届欧洲计算机系统会议论文集（*EuroSys ’20*），1–16。Taft, R.; El-Sayed, N.; Serafini, M.; Lu, Y.; Aboulnaga, A.; Stonebraker, M.; Mayerhofer, R.; and Andrade, F. 2018年。P-store：一种采用预测式资源配置的弹性数据库系统。载于*2018*年国际数据管理会议论文集（*SIGMOD ’18*），205–219。Tsoumakos, D.; Konstantinou, I.; Boumpouka, C.; Sioutas, S.; and Koziris, N. 2013年。利用 TIRAMOLA 为 NoSQL集群自动配置弹性资源。载于*2013*年 *IEEE/ACM*集群、云和网格计算国际研讨会，34–41。Wen, Q.; Gao, J.; Song, X.; Sun, L.; and Tan, J. 2019年。RobustTrend：采用 Huber损失以及结合一阶和二阶差分正则化的时间序列趋势滤波方法。载于*IJCAI 2019*，3856–3862。Wen, Q.; Gao, J.; Song, X.; Sun, L.; Xu, H.; and Zhu, S. 2019年。RobustSTL：一种适用于长时间序列的鲁棒季节性-趋势分解算法。载于*AAAI*人工智能会议（*AAAI*），5409–5416。Wen, Q.; He, K.; Sun, L.; Zhang, Y.; Ke, M.; and Xu, H. 2021年。RobustPeriod：用于鲁棒多重周期检测的时频挖掘。载于*ACM*数据管理国际会议（*SIGMOD*），2328–2337。Wen, Q.; Zhang, Z.; Li, Y.; and Sun, L. 2020年。Fast RobustSTL：针对复杂模式时间序列的高效、鲁棒季节性-趋势分解。载于第*26*届 *ACM SIGKDD* 知识发现与数据挖掘国际会议论文集，2203–2213。Wen, Q.; Zhou, T.; Zhang, C.; Chen, W.; Ma, Z.; Yan, J.; and Sun, L. 2022年。时间序列中的 Transformer：综述。*arXiv*预印本 *arXiv:2202.07125*。Yan, Z.; Lu, J.; Chainani, N.; and Lin, C. 2021年。面向工作负载的自主数据库管理系统性能调优。载于*2021*年 *IEEE* 第*37*届数据工程国际会议（*ICDE*），2365–2368。Zhou, T.; Ma, Z.; Wen, Q.; Wang, X.; Sun, L.; and Jin, R. 2022年。FEDformer：用于长期序列预测的频率增强分解Transformer。*ICML2022*。