A Reading Report About AHPA

邹翔宇 2410833001
陈锦新 2410833003

1. Background

1.1 Pod

Pod 是可以在 Kubernetes 中创建和管理的、最小的可部署的计算单元。Pod 可以被理解成一群可以共享网络、存储和计算资源的容器化服务的集合。在同一个 Pod 里的几个 Docker 服务程序, 好像被部署在同一台机器上, 可以通过 localhost 互相访问, 并且可以共用 Pod 里的存储资源。同一个 Pod 之间的 Container 可以通过 localhost 互相访问, 并且可以挂载 Pod 内所有的数据卷; 但是不同的 Pod 之间的 Container 不能用 localhost 访问, 也不能挂载其他 Pod 的数据卷。

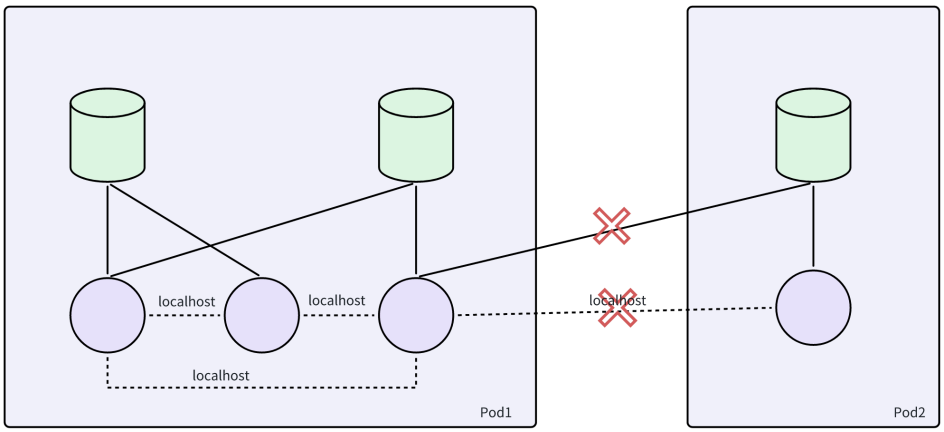

图 1 pod 示意图

1.2 Deployment 和 ReplicaSet

Deployment 的作用是管理和控制 Pod 和 ReplicaSet, 管控它们运行在用户期望的状态中。ReplicaSet 的作用就是管理和控制 Pod, 管控他们好好干活。但是, ReplicaSet 受控于 Deployment。这样层级管理制度是为了实现前后两个版本平滑升级、平滑回滚等高级功能,见图 2。

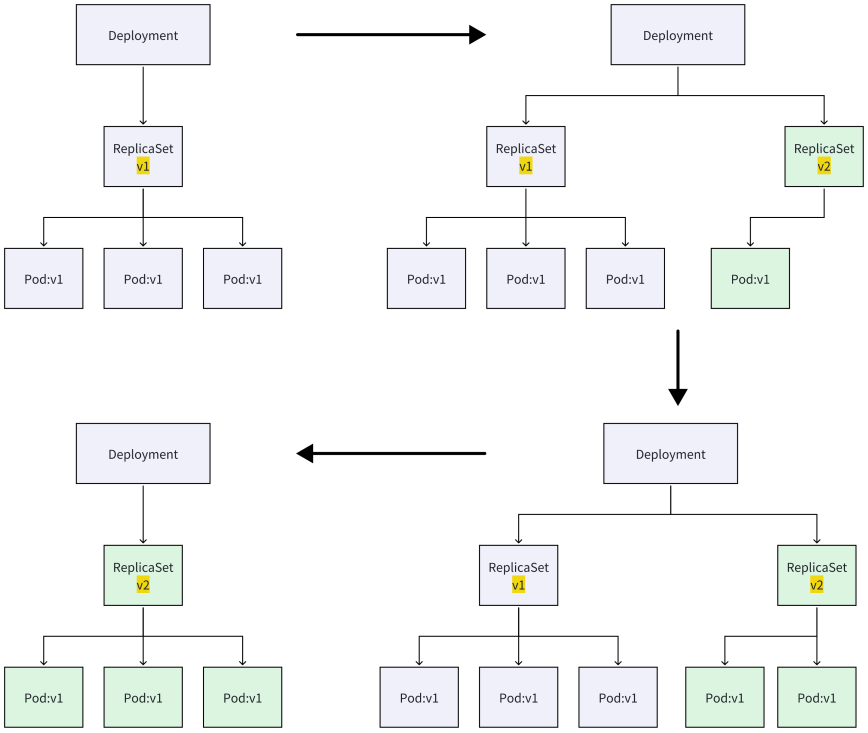

图 2　Deployment 升级策略

1.3　HPA

Horizontal Pod Autoscaling（HPA）是 Kubernetes 中的一种自动缩放机制。它能够自动更新工作负载资源（如 Deployment），通过部署**更多 Pod** 来应对增加的负载，实现水平扩展，以匹配需求。当负载降低且 Pod 数量高于配置的最小值时，它会指示工作负载资源缩减规模。作为 Kubernetes API 资源和控制器实现。资源定义了控制器的行为，控制器在 Kubernetes 控制平面内运行，定期根据观察到的指标（如平均 CPU 利用率、平均内存利用率或指定的其他自定义指标）调整目标（如 Deployment）的期望规模。例如，它会根据这些指标决定是否需要增加或减少 Pod 的数量。

Horizontal Pod Autoscaling（HPA）的自动缩放机制在 Kubernetes 中按以下方式工作：

1. 基于资源指标（如 CPU、内存）

- 指标收集：
  - HPA 控制器会定期从集群中的 Metrics - Server（或其他配置的资源指标采集器）获取 Pod 的资源使用情况，包括 CPU 和内存等指标的使用数据。Metrics-Server 会收集每个 Pod 的实时资源使用信息。
- 指标评估与决策：
  - 对于基于 CPU 利用率的缩放，例如，在 HPA 定义中设置了目标 CPU 利用率为 50%。HPA 控制器会计算所有 Pod 的平均 CPU 利用率。如果平均 CPU 利用率超过了设定的 50% 阈值，并且当前 Pod 副本数小于 HPA 中定义的最大副本数，HPA 就会决定增加 Pod 副本数。

- 假设当前有 3 个 Pod，平均 CPU 利用率达到了 60%，而 HPA 的最大副本数为 10，那么 HPA 可能会决定增加 Pod 副本数，比如增加到 4 个，以分担负载，降低每个 Pod 的 CPU 使用率。相反，如果平均 CPU 利用率低于阈值且副本数大于最小副本数，HPA 会减少副本数。例如，当平均 CPU 利用率下降到 30%，而最小副本数为 2，HPA 可能会将 Pod 副本数减少到 2 个，以避免资源浪费。

2. 基于自定义指标（应用特定指标）

- 自定义指标提供：
  - 如果要根据应用程序自定义的指标（如每秒请求数、队列长度等）进行缩放，需要先在集群中实现自定义指标 API。通常会部署如 Prometheus 等提供自定义指标的服务，并将其与 Kubernetes 集成，使 HPA 控制器能够获取这些自定义指标的值。
- 缩放操作触发：
  - 例如，定义了一个自定义指标 http_requests_per_second（每秒请求数），在 HPA 资源定义中设置了目标值为 100（即当平均每秒请求数达到 100 时进行缩放）。HPA 控制器会从自定义指标 API 获取该指标的值，当该值达到或超过设定目标时，如果当前副本数未达到最大限制，HPA 会增加 Pod 副本数；如果该值低于设定目标且副本数大于最小限制，HPA 会减少副本数。

3. 基于外部指标（集群外部系统提供的指标）

- 外部指标引入：
  - 类似于自定义指标，需要配置集群以支持获取外部指标。这可能涉及设置相关的适配器或服务，将外部系统（如云服务提供商的监控服务）的指标引入到 Kubernetes 集群中，使 HPA 控制器能够访问这些指标。
- 根据外部指标缩放：
  - 例如，如果使用云服务提供商的特定指标（如 AWS 的特定负载均衡器的请求数等），在 HPA 定义中引用该外部指标。当外部指标的值满足 HPA 中设定的缩放条件（如达到一定的请求数阈值）时，HPA 会相应地调整 Pod 副本数，以适应外部系统的负载变化。

通过以上机制，HPA 能够根据不同类型的指标自动调整 Pod 副本数，使应用程序能够根据实际负载情况动态地分配资源，提高应用的性能、可用性和资源利用率。

2. HPA Simple Survey

2.1 Auto-scaling Techniques[1]

2.1.1 Threshold-Based Rules

- 该技术涉及定义一些规则，这些规则会根据特定性能指标（如 CPU 使用率）的预定义阈值来触发伸缩操作。
- 这些规则实施起来较为简单，对于工作负载可预测的应用程序通常较为有效。
- 然而，对于需求波动的应用程序来说，基于阈值的规则可能很难进行有效配置，因为阈值选择不当可能会导致系统振荡和资源过度配置。

2.1.2 Reinforcement Learning (RL)

- 强化学习技术旨在通过试错来学习针对不同应用状态的最优伸缩操作。
- 强化学习方法不需要预先定义应用程序模型；相反，它们会在线学习模型并适应不断变化的条件。
- 尽管强化学习具有适应性且无需模型要求，但它可能在计算上开销较大，而且往往学习阶段较长，这使得它不太适合在生产系统中进行实时决策。

2.1.3 Queuing Theory （QT）

- QT 通常在分析阶段用于估算性能指标，如队列长度和请求的平均等待时间。
- 它涉及将系统建模为由队列和服务器组成的网络，然后运用数学公式或模拟来预测其行为。
- QT 为理解系统性能提供了理论基础，但在处理具有动态工作负载和多种资源类型的复杂场景时可能会存在困难。

2.1.4 Control Theory (CT)

- 控制论通过调整资源分配以维持期望的性能水平（例如使 CPU 负载接近目标值），将控制系统应用于自动伸缩过程。
- 可以使用不同类型的控制器，从较为简单的固定增益控制器（如比例-积分-微分控制器，PID）到更为复杂的自适应控制器（如模糊控制器和模型预测控制器）不等。
- CT 为自动伸缩提供了一个稳健的框架，但需要对应用程序进行仔细建模，并且对模型的不准确性较为敏感。

2.1.5 Time Series Analysis

这种方法侧重于分析历史性能数据，以识别反复出现的模式，并有可能预测未来的资源需求。诸如移动平均、指数平滑、自回归以及机器学习模型等技术通常会被使用。时间序列分析对于工作负载模式可预测的应用程序可能非常有效，但在适应突发变化或意外事件方面可能会存在困难。

2.2 Auto-scaling in Kubernetes

- Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration[2]
  - 探讨了 HPA 的基本机制，并分析了其在不同条件下的性能。它详细介绍了 Kubernetes 的架构，着重强调了参与 HPA 操作的组件，如 “kube-controller-manager”、“kube-apiserver” 和 “cAdvisor”。该部分内容强调了 Kubernetes 资源指标（Kubernetes Resource Metrics，KRM）和 Prometheus Custom Metrics 之间的差异，这两者是 HPA 用于做出扩缩决策的关键指标来源。KRM 依赖于定期采集资源使用数据，而 PCM 通过普罗米修斯提供了更动态、更灵活的监控能力。该部分内容研究了采集周期对 HPA 性能的影响，并比较了在各种负载场景下 KRM 和 PCM 的有效性。值得注意的是，该部分内容观察到，与 KRM 相比，PCM 对工作负载波动的响应速度更快，从而导致更积极的扩缩操作。然而，这种响应速度是以在扩缩事件期间由于新创建的 Pod 准备就绪所需时间而导致失败请求增加为代价的。
- Graph-PHPA: Graph-based Proactive Horizontal Pod Autoscaling for Microservices using LSTM-GNN[3]
  - 基于图的用于微服务的主动水平 Pod 自动扩缩（使用 LSTM-GNN）介绍了一种全新的 HPA 方法，该方法结合了机器学习技术以实现主动扩缩。它利用长短期记忆（Long Short-Term Memory，LSTM）网络和图神经网络（Graph Neural Networks，GNNs）的组合来预测未来的工作负载模式和资源需求。所提出的 Graph-PHPA 模型包括两个阶段：
    - 工作负载预测：一个基于历史工作负载数据训练的 LSTM 模型预测应用程序中每个微服务的未来工作负载
    - 资源使用预测：一个依据预测的工作负载和应用程序的图结构（捕捉微服务之间的依赖关系）的 GNN，预测每个微服务所需的虚拟 CPU（vCPU）分配。

  该部分内容强调了 Graph-PHPA 主动特性的优势，使其能够预测工作负载变化并预先分配资源，从而与传统 HPA 的被动响应方法相比，最大限度地减少响应时间并有可能降低资源消耗。
- **Graph-PHPA 相对于 HPA 的改进：**
  - 主动扩缩：Graph-PHPA 采用预测模型来预测未来工作负载变化并主动调整资源分配，这与传统 HPA 的被动扩缩机制不同，传统 HPA 是根据当前资源使用阈值做出反应。这种主动性旨在提高响应时间并有可能提高资源利用效率。

- 工作负载模式学习：LSTM 网络的使用使 Graph-PHPA 能够学习工作负载模式中的复杂时间依赖关系，与仅仅依赖即时资源使用指标相比，有可能做出更准确的预测。
- 微服务依赖关系建模：通过纳入 GNN，Graph-PHPA 明确地对应用程序内微服务之间的关系进行建模。对服务依赖关系的这种认知有助于更智能的资源分配，确保扩缩决策考虑到微服务的相互关联性质。

3. AHPA

3.1 现有方法存在的问题

现有方法主要包括固定数量实例、HPA（Horizontal Pod Autoscaling）和 CronHPA，它们存在的问题如下：

- 固定数量实例
  - 资源浪费：在业务需求低谷期，由于实例数量固定不变，会导致资源大量浪费。
- HPA（Horizontal Pod Autoscaling）
  - 响应滞后：在需求发生变化后才调整实例数量，对需求波动的响应存在滞后性，可能导致处理无效、服务质量差，甚至应用被终止。
- CronHPA
  - 依赖专家经验：需要专家经验手动设置缩放计划，这种方式可能不准确、不灵活，并且会耗费大量人力资源。
  - 缺乏动态调整能力：与 HPA 类似，它也不能动态地根据实际业务需求进行弹性调整，难以适应复杂多变的业务场景。
- 总体问题
  - 无法弹性处理需求波动：上述三种方法均无法有效地解决业务需求的弹性波动问题，在面对具有周期性或突发变化的业务时，无法实现资源的高效利用和服务的稳定保障。

3.2 AHPA 的具体方案

AHPA[4]算法的主要工作流程有以下几个部分,见图 3：

1. Collect Historical Data
2. Data Preprocessing
3. Future Workload Forecasting
4. Performance Model Training
5. Scale Plan Generation
6. Target QoS level & Satisfaction probability

接下来，我们将具体来介绍这几个流程。

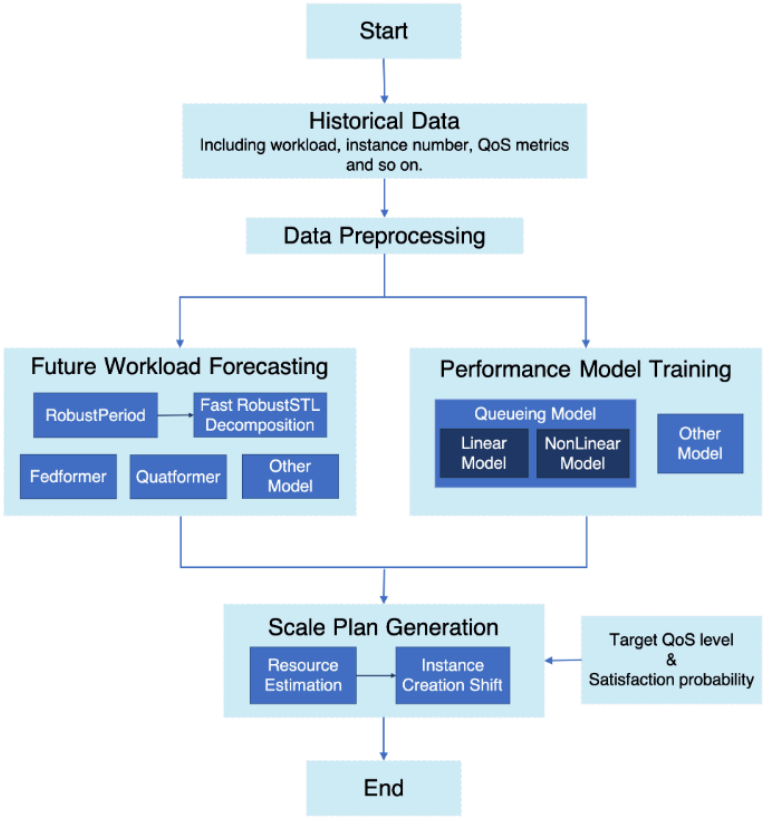

图 3　AHPA 工作流程

3.2.1　Historical Data

首先, 收集包括工作量(workload)、实例数量(instance number)和服务质量指标(QoS metrics)等历史数据。这些数据将作为后续步骤的基础。

3.2.2　Data Preprocessing

对收集到的历史数据进行预处理，为后续的预测和模型训练做准备。

3.2.3　Future Workload Forecasting

在阿里云服务系统中的预测过程存在许多复杂挑战。

- 数据丢失以及噪声数据：在分布式云节点的运行过程之中，可能会出现系统宕机或者其他意外，就会造成数据的丢失或者产生噪声数据，因此设置阈值判断数据有效性，并且在某些情况下需要合适的方法来补充缺失数据和归一化数据规模。
- 有限的数据规模: Kubernetes 中的指标数据通常使用 Prometheus 存储, 出于成本和效率的权衡, 一般业务数据存储周期为 7 天，7 天的数据量作为训练集太小，训练出的机器学习或深度学习模型通常准确性较差。如何利用有限的历史数据有效预测未来业务量是值得探讨的问题。

- 高复杂性数据：一般来说，用户需求频繁变化，这极大地增加了数据的复杂性。例如，数据可能具有多周期等复杂特性。复杂的数据对算法模型准确预测的能力有更高要求。

AHPA 设计了一种基于稳健分解的统计方法作为主要预测方案来应对上述挑战，并满足高预测延迟要求。首先，使用基于 MODWT 的 RobustPeriod 来检测输入时间序列是否具有周期性及其周期长度。然后根据周期检测结果，对于周期性数据，采用 RobustSTL 将输入时间序列分解为趋势、季节性（周期成分）和残差项；对于非周期性数据，采用 RobustTrend 将输入时间序列分解为趋势和残差项。图 4 是 AHPA 的时间序列分解模块。

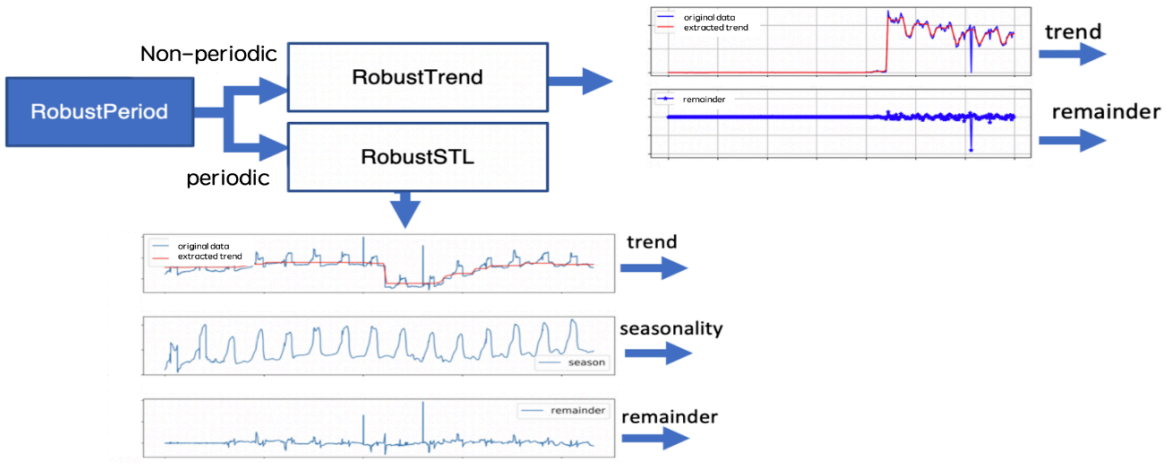

图 4　The robust time series decompostition for forecasting module in AHPA

- RobustPeriod，基于 MODWT（最大重叠离散小波变换）来解耦多个周期性，用于检测输入时间序列是否具有周期性及其周期长度。当需要判断数据是否存在周期性特征时使用。通过这种方法，可以帮助确定后续处理中对数据应采用的分解策略。
- RobustSTL，用于将输入时间序列分解为趋势、季节性（周期成分）和残差项。它是针对周期性数据所采用的处理方法。在 RobustPeriod 检测出数据具有周期性特征后，使用 RobustSTL 对周期性数据进行进一步分解，以便更好地理解数据的组成部分，为后续的预测和分析提供基础。
- RobustTrend, 用于将输入时间序列分解为趋势和残差项。它是针对非周期性数据所采用的处理方法。当 RobustPeriod 检测出数据不具有明显的周期性特征时，使用 RobustTrend 对非周期性数据进行分解，以便更好地理解数据的趋势变化和残差项，为后续的预测和分析提供基础。

文中提到的分解方法会将时间序列分解为 trend、seasonality 和 residual terms，它们的作用如下：

- trend
  - 趋势反映了数据在较长时间跨度内的总体走向。它能够展示出数据是处于上升、下降还是相对平稳的状态。

- 通过识别趋势，可以帮助预测未来数据的大致走向。例如，在业务场景中，如果数据的趋势是持续上升的，那么可以预期未来业务量可能会继续增长，从而可以提前规划资源，如增加服务器数量等，以满足未来业务需求。

- seasonality
  - seasonality 代表了数据中重复出现的周期性波动模式。这些波动通常与时间相关，例如按天、按周、按月等周期性循环。
  - 了解季节性有助于预测在特定时间段内数据的波动情况。以在线购物为例，可能在每年的特定节日期间（如双十一）业务量会出现高峰，这种季节性的特征可以帮助商家提前准备库存、安排物流和调配客服人员等资源，以应对业务高峰。
- residual terms
  - 残差项是在从原始数据中去除趋势和季节性成分后所剩下的部分。它包含了无法由趋势和季节性解释的随机波动和噪声。
  - 尽管残差项代表了数据中的不确定性和随机性，但对其进行分析仍然有意义。它可以帮助发现数据中的异常情况，例如突发的业务变化（如因突发事件导致的业务骤降或骤升），从而可以及时采取应对措施，避免业务受到较大影响。

从数学上来说，robust forecasting 将时间序列分解 y_t 为 trend(τ_t), period item($s_{i,t}$), residual item(r_t)，如式 1.

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

式 2 是用于预测下一个时刻y_{t+1}的数据值。

$$\begin{aligned} y_{t+1} = &\sum_{i=1}^{m}(s_{i,t}) + \text{ExponentialSmoothing}(\tau_t) \\ &+ \text{QuantileRegressionForest}(r_t) \end{aligned} \tag{2}$$

3.2.4 Performance Model Training

该部分旨在通过对历史数据进行训练，构建一个性能模型，使其能够准确预测未来的业务性能指标，如工作负载、资源需求等，从而为资源的合理分配和调度提供依据。AHPA 中主要采用运筹学中排队论的方法，包括两种不同的排队模型，M/M/1 队列和 M/M/c 队列。

- M/M/1，是单服务台排队模型，即系统中只有一个服务台为顾客提供服务。在文中将每个 Pod 看作一个独立的服务台，每个服务台具有相同的处理速率。$\text{AvgRT(M/M/1)} = (\mu - \text{QPS})^{-1} + \text{otherlatency}$其平均等待时间与到达率和服务率的差值有关，当到达率接近服务率时，等待时间会急剧增加。
- M/M/c，是多服务台排队模型，系统中有 c 个相同的服务台并行工作，且这些服务台共享一个公共缓冲区。平均等待时间公式为$\text{AvgRT(M/M/c)} = f(\text{QPS}, \mu, N) + \text{otherlatency}$。

在 M/M/c 多服务台排队模型场景中，平均响应时间不再像 M/M/1 模型那样简单地由服务率和到达率的关系决定，而是要通过更为复杂的函数，综合考虑到达率（QPS）、服务率（μ）以及相关数量因素（N）来计算。在处理高并发请求时，M/M/c 模型由于多个服务台并行工作，其平均等待时间相对 M/M/1 模型可能会更短，系统性能更优。

- 适用场景
  - M/M/1：当 Pod 数量相对较少，且业务请求量相对较小时，M/M/1 模型可能表现较好，因为此时单个 Pod 足以应对请求，且模型简单易于理解和计算。在实际应用中，如果业务的并发度不高，资源需求相对稳定，M/M/1 模型可以提供较为准确的性能评估和资源配置建议。
  - M/M/c：当业务请求量较大，且 Pod 数量可灵活调整时，M/M/c 模型更具优势。它能够充分利用多个服务台的并行处理能力，有效降低请求的平均等待时间，提高系统的整体性能。例如在一些大型互联网应用中，面对高并发的用户请求，采用 M/M/c 模型可以更好地优化资源配置，提升用户体验。

3.2.5 Scale Plan Generation

- 基于预测与模型生成决策：
  - 系统会依据未来工作量的预测结果以及性能模型来生成最终的扩展决策。这里的未来工作量预测能够预估业务需求的变化情况，而性能模型则帮助确定满足相应业务性能指标（比如平均响应时间 RT 或者 CPU 使用率等）所需要的资源状况，二者结合为生成合理的扩展计划提供基础。
  - 扩展计划涵盖了 Pod 数量以及对应的增减时间信息，明确在什么时间点应该增加或者减少多少个 Pod，以此来动态调整资源，适应业务的波动。
- 时间延迟问题与应对方法：
  - 首先，在预测环节，虽然可以根据相关指标要求算出满足客户需求的 Pod 数量，但 Pod 启动是存在开销时间的，例如启动过程中的初始化配置、加载资源等操作都需要花费时间，这个时间通常不能忽略。
  - 这就导致在添加资源时，系统会面临时间延迟问题，即按照实时需求去精准扩展 Pod 数量，实际可用的 Pod 并不能马上投入使用，存在一个时间差，并且部署吞吐量也是有限的，也就是单位时间内能够完成部署并投入使用的 Pod 数量存在上限。
  - 为应对这些约束，采用了改进的预测偏移算法。通过提前规划来尽量抵消启动延迟带来的影响，保障业务在需要资源时能及时有足够可用的 Pod 来处理请求。
- 操作频率限制及计划整合问题：
  - 出于稳定性方面的考虑，频繁地进行 Pod 数量的增减操作并不是一个好的选择。因为过于频繁的操作可能会导致系统资源频繁调配、状态不稳定，甚至影响业务的正常运行。
  - 通常会将操作频率限制设定为每 3 分钟一次或者最多每 5 分钟一次。而在这样设定的时间间隔内，如何合理地整合调度计划就成了一个值得深入探究的问题。比如如何根据业务需求变

化趋势，在允许的操作时间间隔内，选择最合适的时间点去执行 Pod 数量的调整，以达到既能满足业务性能要求，又能保证系统整体稳定的效果。

3.3 AHPA 的系统架构

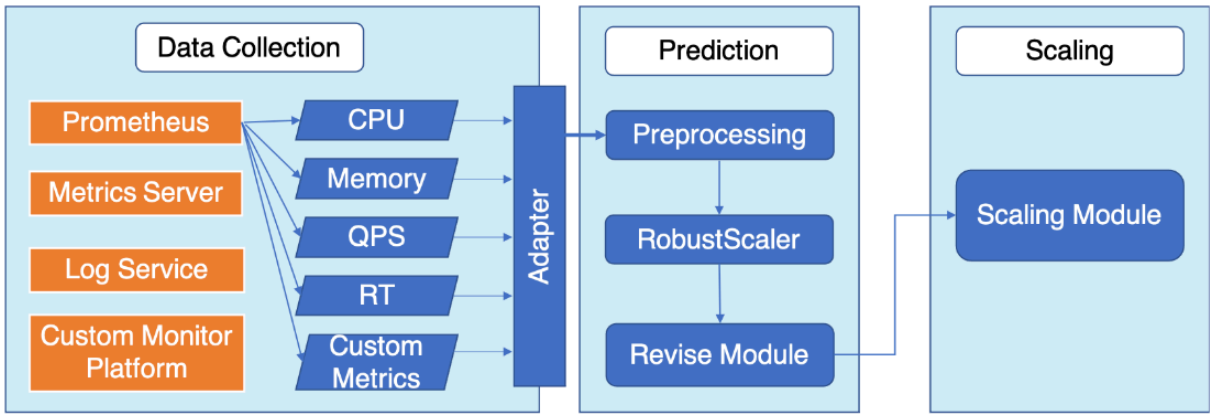

图 5　AHPA 系统架构

1. 数据采集（Data Collection）

- 数据来源：
  - Prometheus：这是一个常见的开源监控和告警工具，能够提供多种系统指标数据。
  - Metrics Server：用于收集 Kubernetes 集群中的资源使用度量数据，如 CPU、内存等。
  - Log Service：负责记录系统运行时产生的日志数据，这些数据对于分析系统状态和故障排查非常重要。
  - Custom Monitor Platform：自定义监控平台，可以收集特定业务相关的自定义指标数据。
- 采集数据类型：
  - CPU：中央处理器的使用情况数据。
  - Memory：系统内存的使用情况数据。
  - QPS（Queries Per Second）：每秒查询量，反映系统的负载情况。
  - RT（Response Time）：响应时间，用于衡量系统处理请求的速度。
  - Custom Metrics：自定义指标，满足特定业务需求的数据。

2. 预测（Prediction）

- 数据预处理（Preprocessing）：
  - 从数据采集部分获取到各种数据后，首先要进行预处理。预处理的目的是清理、转换和规范化数据，使其适合后续的预测模型使用。
- RobustScaler：

- 经过预处理的数据进入 RobustScaler。RobustScaler 采用稳健的时间序列分析方法，对未来的工作负载进行预测。它能够处理数据中的异常值和噪声，提供较为准确的预测结果。

- 修正模块（Revise Module）：
  - 对稳健扩展器的预测结果进行修正。这一模块可能会根据历史数据、当前系统状态和其他相关因素，对预测结果进行调整和优化，使其更加符合实际情况。

3. 扩展（Scaling）

- 扩展模块（Scaling Module）：
  - 根据预测模块提供的未来工作负载预测结果，扩展模块决定是否需要对系统资源进行扩展或收缩。例如，如果预测到未来工作负载将大幅增加，扩展模块会触发增加 Pod 数量的操作，反之则减少 Pod 数量。

整个系统架构通过数据采集收集系统运行数据，经过预测部分对未来工作负载进行预测和修正，最后由扩展模块根据预测结果决定资源的扩展或收缩，从而实现自适应的水平 Pod 自动扩展功能，以满足系统在不同负载情况下的资源需求。

3.4 AHPA 的优势

- 动态资源调整
  - 准确预测业务需求：使用基于稳健分解的统计方法（如 RobustPeriod、RobustSTL、RobustTrend 等算法）以及深度学习模型（如 FEDformer、Quatformer 等，适用于数据充足场景）对未来业务进行预测，能够提前感知业务变化趋势，解决了现有方法无法动态调整 POD 资源以适应业务波动的问题，避免了资源浪费或不足的情况。
- 优化资源分配
  - 性能模型训练：通过采用排队理论中的不同模型（如线性关系的并行 M/M/1 队列和具有公共缓冲 pod 的 M/M/c 队列），根据业务 QPS、Pod 数量、Pod CPU 使用率等指标，模拟指标度量与所需 Pod 数量之间的关系，找到满足平均响应时间（RT）要求的最小 Pod 数量，从而优化资源分配，提高资源利用率。
- 自动化与智能化
  - 自动生成缩放计划：根据预测的工作量和性能模型，自动生成包括 Pod 数量和增减时间的缩放计划，无需像 CronHPA 那样依赖人工手动设置，提高了调整的准确性和灵活性，节省了运维成本。
- 智能弹性模块：在弹性指标部分，智能弹性模块结合前瞻性预测模块和降级保护模块，能够根据预测结果和保障稳定性的需求，自动输出资源预测，进一步优化资源分配和调整策略。

3.5 AHPA 的不足

- 映射关系的局限性: AHPA 算法核心部分目前主要使用基于稳健分解的时间序列预测模块和队列理论进行性能建模，对于从业务工作量到所需 Pod 数量的映射关系，主要依赖于队列理论。在面对复杂多样的业务场景时，这种单一的理论基础可能无法全面准确地描述和处理各种情况，存在一定的局限性。
- 应对大规模复杂业务需求的挑战: 随着云服务的蓬勃发展，越来越多的大型需求不断涌现，业务场景变得更加复杂多样。AHPA 在处理一些极端复杂或特殊的业务场景时，可能面临适应性方面的挑战，需要进一步提升其灵活性和通用性，以更好地满足不同业务场景下的高效资源管理需求。

4. 相关工作

4.1 Smart HPA

2024 年，AHMAD H 等人提出了 Smart HPA[5]，Smart HPA 提出一种融合集中式和分散式架构风格的**层次化架构**,如图 6，包括微服务管理器（Microservice Manager）、微服务容量分析器（Microservice Capacity Analyzer）和自适应资源管理器（Adaptive Resource Manager）三个主要组件。该架构旨在充分发挥两种架构风格的优势，应对微服务架构中自动缩放操作管理的局限性，特别适用于资源受限环境下的微服务资源管理。

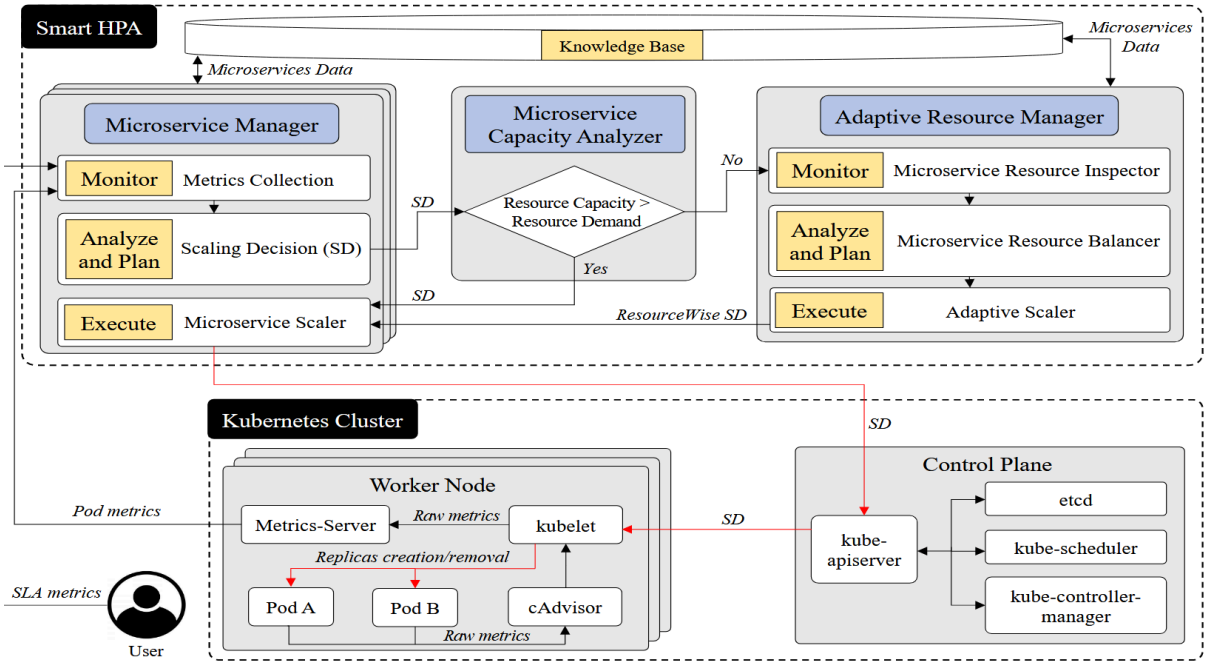

图 6　Smart HPA 架构

Smart HPA 微服务管理器为每个微服务独立收集和分析数据，采用静态阈值策略确定期望副本数并检测违规以做出缩放决策。微服务容量分析器评估资源需求与容量，当资源需求超容量时触发自适应资源管理器。自适应资源管理器通过资源高效启发式算法，识别资源过剩和不足的微服务，在它们之间转移资源，调整资源容量和期望副本数，以确保每个微服务的资源需求得到满足。通过资源高效启发式算法，Smart HPA 可以实时监测微服务的资源需求变化，并动态地在微服务之间进行资源平衡。当某个微服务的资源需求突然增加时，能够迅速从其他有剩余资源的微服务获取所需资源，而当资源需求降低时，也能及时释放资源给其他需要的微服务。这使得系统在资源受限环境下能够保持良好的性能和稳定性。

4.2 DeepScaling

Wang 等人提出了 DeepScaling[6]，一种用于大规模云系统中微服务自动缩放的创新框架，通过综合运用多种技术，DeepScaling 在维持稳定 CPU 利用率和保障服务质量方面表现出色，为云原生环境中的自动缩放提供了有效解决方案。

4.2.1 精准的工作量预测

利用**时空图神经网络（STGNN）**建模服务调用图和多变量关系，综合考虑多种工作量指标（如 RPC 请求、文件 I/O 等）间的相互依赖关系，能准确预测不同服务的工作量。实验表明，相比 N - beats 和 Transformer 等模型，DeepScaling 在预测突发工作量变化时表现更优，预测误差显著降低，为后续资源调整提供可靠依据。

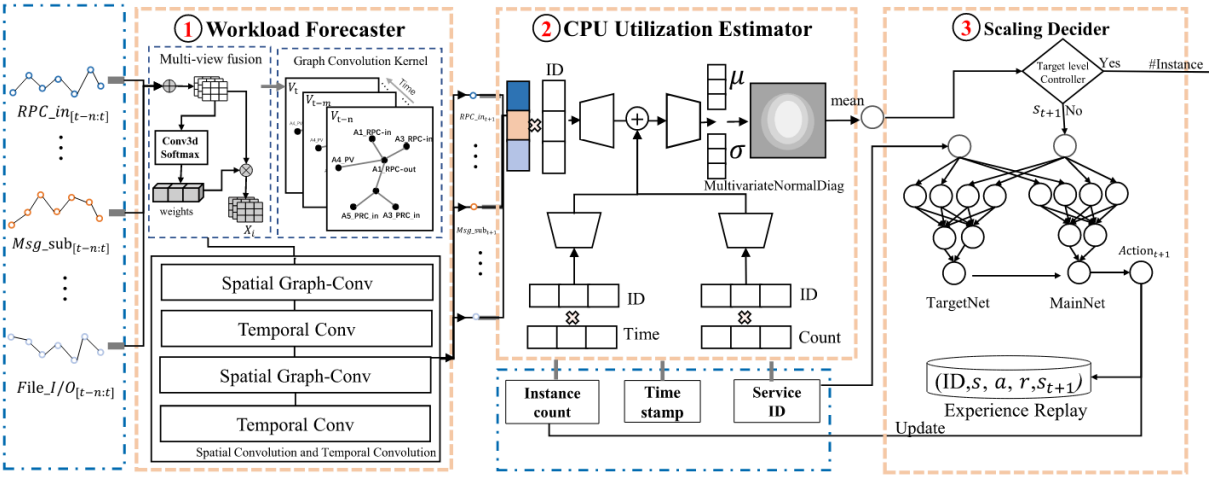

图 7　DeepScaling 处理流程

4.2.2 精确的 CPU 利用率估计

基于**深度概率回归网络**，结合多维工作量指标与辅助特征（服务 ID、时间戳、实例数量），通过特征嵌入和深度神经网络架构，准确估计 CPU 利用率。与线性回归、支持向量机和决策树回归等传统方法相比，DeepScaling 在处理复杂非线性数据时优势明显，估计误差大幅降低，可有效避免资源分配不当。

4.2.3 高效的自动缩放决策

采用改进的**深度 Q 网络（DQN）**模型，结合强化学习，根据 CPU 利用率估计和预测工作量确定服务资源需求。通过合理定义状态空间、动作空间和奖励函数，快速找到最优资源配置。在实际应用中，能快速调整实例数量，使服务 CPU 利用率稳定在目标水平，提高资源利用率，减少资源浪费。

参考文献

[1] LORIDO-BOTRAN T, MIGUEL-ALONSO J, LOZANO J A. A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments[J/OL]Journal of Grid Computing, 2014, 12: 559-592. https://api.semanticscholar.org/CorpusID:5746950

[2] NGUYEN T T, YEOM Y J, KIM T, 等. Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration[J/OL]Sensors (Basel, Switzerland), 2020, 20. https://api.semanticscholar.org/CorpusID:221240122

[3] NGUYEN H X, ZHU S, LIU M. Graph-PHPA: Graph-based Proactive Horizontal Pod Autoscaling for Microservices using LSTM-GNN[C/OL]//2022 IEEE 11th International Conference on Cloud Networking (CloudNet)IEEE, 2022: 237-241. http://dx.doi.org/10.1109/CloudNet55617.2022.9978781. DOI:10.1109/cloudnet55617.2022.9978781

[4] ZHOU Z, ZHANG C, MA L, 等. AHPA: Adaptive Horizontal Pod Autoscaling Systems on Alibaba Cloud Container Service for Kubernetes[C/OL]//AAAI Conference on Artificial Intelligence2023. https://api.semanticscholar.org/CorpusID:257378203

[5] AHMAD H, TREUDE C, WAGNER M, 等. Smart HPA: A Resource-Efficient Horizontal Pod Auto-Scaler for Microservice Architectures[C/OL]IEEE, 2024: 46-57. http://dx.doi.org/10.1109/ICSA59870.2024.00013

[6] WANG Z, ZHU S, LI J, 等. DeepScaling: Autoscaling Microservices With Stable CPU Utilization for Large Scale Production Cloud Systems[J/OL]IEEE/ACM Transactions on Networking, 2024, 32: 3961-3976. https://api.semanticscholar.org/CorpusID:270186232