# 一种基于工作量的 **Serverless** 计算自动伸缩算法的设计与实现

摘 要

Serverless 计算作为一种新型的计算模式，在云计算和边缘计算领域被广泛应用。本研究针对 Serverless 计算的自动伸缩问题，设计了一种主要基于 ARIMA 模型的的自动伸缩算法，并对比了其他常见预测模型，对其性能进行了全面的比较评估。该算法结合了负载预测和自动伸缩策略，能够根据应用程序的负载变化动态调整计算资源。并且在基础算法基础上引入了自动选择最优模型超参数的功能，更利于实际应用。通过引入多种预测模型（如 MA、VAR、ARIMA 和 Prophet 模型）并对比和改进，相较于目前主流的基于阈值的伸缩，我们的算法提高了负载预测的准确性，实现了对模型的自动训练、动态调整和优化。

在实现算法部署的过程中，我们在 Kubernetes 集群上设计了一种基于 Prometheus 的监控方案，此方案能够收集 Serverless 应用程序的工作负载数据，动态地根据历史数据，预测下一时间段的负载，进而通过 HPA 控制器自动调整 Serverless 函数的副本数量。

实验结果表明，这种智能化的自动伸缩算法可以有效预测副本数量，提高 Serverless 应用程序的资源利用率，减少计算资源浪费，同时保证应用程序的稳定性和可靠性。

关键词 Serverless 云计算 时序预测 自动伸缩 ARIMA

**Design and Implementation of Workload-based**

**Auto-scaling Algorithm for Serverless Computing**

**ABSTRACT**

Asa nascentcomputing paradigm, Serverless computinghasgained tractioninthedomains of cloud and edge computing. This study presents a workload-oriented auto-scaling algorithm designed specifically for Serverless computing. This algorithm, marrying load prediction and auto-scaling policy, can dynamically allocate computational resources in accordance with application load. Leveraging a range of models such as MA, VAR, ARIMA, and Prophet, it seeks to enhance the precision of load prediction. It self-trains and dynamically scales computational resources, continuously optimizing models through data obtained via service monitoring. When implemented on a Kubernetes cluster, a Prometheus-based monitoring solution is designed in this paper. This solution aggregates metric data from Serverless applications and leverages statistical model training features to forecast future load, subsequently utilizing the HPA controller to auto-adjust the replica count of Serverless functions. The findings illustrate that the proposed smart algorithm can effectively estimate replica counts, augmenting Serverless application resource utilization, curbing computational resource wastage, and bolstering application stability and reliability.

**KEYWORDS** Serverless Cloudcomputing Timeseriesprediction Autoscaling ARIMA

II

## 第一章 引言

### **1.1** 背景介绍

#### 1.1.1 Serverless 计算概述

Serverless 计算是云计算的一种新模式，它可以动态地扩展和缩放计算资源，按需自动调配计算资源来运行业务代码。不同于传统的基础设施即服务（IaaS）和平台即服务（PaaS）模式，Serverless 模式管理和操作底层基础设施和服务器，使开发者可以专注于业务逻辑和代码的开发，无需管理服务器和基础架构。

在 Serverless 模式下，资源由事件驱动自动调配，开发者只需编写并部署业务逻辑的代码（往往为小段的功能代码），并为其指定触发事件，云服务商会在事件发生时自动执行对应的代码段。开发者无需提前规划资源或服务器数量，也无需维护服务器及其运营。这大大降低了开发和维护成本，使开发者可以更专注于产品创新。

主流云计算服务商都提供了 Serverless 产品和服务，如 AWS Lambda，Azure Functions，Google Cloud Functions 等。此类产品通过事件驱动执行无服务器代码段，并能够自动适配计算资源，为 Serverless 应用快速扩缩容提供支撑。目前，Serverless 计算正快速发展，在云原生应用、微服务架构以及边缘计算等领域有着广泛的应用。

作为云计算的一种新模式，Serverless 计算具有动态扩展、按需计费、自动运维等显著特点，大大降低了开发和维护成本，为应用的快速开发和创新提供了很好的支撑。其作为云计算发展的方向之一，值得持续关注和研究。

##### **1.1.2** 自动伸缩的重要性与挑战

自动伸缩是 Serverless 计算模式下的一项关键功能。它能够动态地按需扩展或缩减计算资源，以适应工作负载的变化，从而提高资源利用率并优化成本。

自动伸缩的重要性在于，在云环境下，工作负载往往是不可预测和突发的。在没有启用自动伸缩时，只能依赖于管理员静态预先配置计算资源来应对业务流量。由于无法准确预测工作负载的模式和峰值，静态预配计算资源很可能导致资源浪费，或无法满足工作负载的需求。动态扩展资源能够在负载增加时迅速提供更多的计算能力，而在需求下降时，也能够快速释放多余的资源，避免资源闲置。这可以显著提高整体资源利用率，并实现按需付费。

自动伸缩也面临一些技术挑战。首先，自动伸缩系统需要准确而高效的负载监控和预测机制，以便能够及时发现负载变化并作出相应调整。其次，资源调度和扩展需要考虑成本优化和服务质量之间的平衡。此外，频繁的资源扩展和释放也会带来性能开销，因此需要进行优化以实现平滑的伸缩。最后，与传统的静态资源池相比，动态变化的资源拓扑也增加了运维的复杂性。

即便面临挑战，自动伸缩已经成为 Serverless 架构的关键。因此，未来工程师需要开发更智能的监控预警手段、更高效的资源调度算法以及更平滑的伸缩机制，以进一步发挥自动伸缩的优势。这也是 Serverless 计算进一步发展的重要方向之一。

### **1.2** 研究目的与意义

本研究的目的是提出一种适用于 Serverless 计算的自动伸缩算法，它综合了多种可选算法, 通过负载预测和资源调度策略实现计算资源的动态扩容和缩容, 以满足工作负载的要求, 保证 Serverless 应用的高性能、高可靠。

展开来说，本研究具有如下具体目标：

1. **实现应用自动伸缩的智能化。**不同于现有的阈值触发式自动伸缩机制, 本研究通过采用时间序列模型进行短期负载预测, 实现了基于预测的自动伸缩决策, 更加智能和灵活。随着实际负载数据积累, 模型也可以动态优化和调整。
2. **提高应用的资源利用率和运行稳定性。**通过负载预测机制, 可以更加精确地预测工作负载的变化, 并相应调整计算资源, 避免资源浪费或不足, 保证服务质量。从而可以降低 Serverless 应用的运行成本, 并提高其可靠性。
3. **提供实际平台上的部署和运维的参考实现。**通过设计一套基于 Prometheus 监控和预警机制, 收集 Serverless 应用的监控指标, 并将数据输入预测模型进行负载预测和自动伸缩决策, 这为 Serverless 应用在 Kubernetes 上的实施提供了参考案例。

本研究希望在实现算法的同时提供可落地的技术方案，为其进一步推广应用提供了理论和技术支撑。

## 第二章 相关工作与理论基础

Serverless 计算是一种无服务器的计算模型，它允许开发者构建和运行应用程序而无需关注基础设施。在 Serverless 架构中，第三方服务（如云提供商）负责管理服务器、网络和其他资源。这样，开发者可以专注于编写应用程序逻辑，而无需担心基础设施的维护和扩展。本章节将介绍 Serverless 计算的基本原理和特点，以及相关的研究工作。

### **2.1 Serverless** 计算的基本原理与特点

从用户的角度来看，Serverless 计算是一种弹性、按需付费、无状态的计算模型。

1. **自动弹性伸缩**：Serverless 应用程序可以根据负载自动调整资源，从而实现高效的资源利用率和低延迟。
2. **按需付费**：与预先分配资源的传统计算模型不同，Serverless 计算按实际使用的资源付费，降低了成本。
3. **无状态性**：Serverless 函数通常是无状态的，这意味着它们不会存储任何关于之前请求的信息。这使得 Serverless 函数易于扩展和管理。

而从研发者的角度，Serverless 计算是一种基于事件驱动的、无服务器的计算模型。

在云服务平台实现 Serverless 计算的方式各异。常见的实现载体包括容器化（Containerization）、虚拟化（Virtualization）。例如，AWS Lambda 采用基于 KVM 的虚拟化技术[1]。

OpenFaaS（Open Function as a Service）是一个开源的 Serverless 计算平台，它允许开发者在 Kubernetes 集群上轻松部署和管理函数。Kubernetes 是一个用于自动部署、扩展和管理容器化应用程序的开源平台。

OpenFaaS 通过以下组件实现 Serverless 计算[2]：

1. **Gateway**：负责处理请求、调度函数、管理函数生命周期，以及提供 API 和仪表板。
2. **Function Watchdog**：作为每个函数的入口点，用于将传入的 HTTP 请求转发到函数。
3. **Function Pods**：包含函数代码和依赖的容器，它们根据负载自动扩展。

在 Kubernetes 集群上部署 OpenFaaS 具有以下优点：

1. **原生集成**：OpenFaaS 利用 Kubernetes 的功能，如自动扩展、滚动更新和自动恢复。
2. **跨平台支持**：Kubernetes 支持多种云提供商和操作系统，这意味着 OpenFaaS 可以在多个平台上运行。
3. **可观测性和监控**：OpenFaaS 可以与 Kubernetes 监控和日志记录工具（如 Prometheus 和 Grafana）集成

### **2.2 Serverless** 计算与自动伸缩方法概述

自动伸缩是一种在计算资源需求变化时，动态调整计算资源分配的方法。现有的自动伸缩方法包括基于单一指标伸缩、手动收缩、根据历史定时伸缩和多指标阈值收缩等。

现有这些方法各自有其优点，但在实际应用中均有局限性。

1. **基于单一指标伸缩**：该方法根据某个指标，如 CPU 使用率或内存使用率等，来调整计算资源。缺点在于，弹性调整速度较慢，可能无法适应突发流量，导致服务性能下降。此外，如果所选指标不准确或阈值设置不合理，可能会导致资源浪费或不足。
2. **手动收缩**：在流量突发时，手动扩展计算资源以满足需求。缺点在于，这种方法响应慢，危险性高，因为在高峰期可能无法及时扩展资源。此外，手动收缩无法实现精细化操作，可能导致资源使用效率低下。
3. **映射历史伸缩**：通过观察周期性，复制历史的收缩曲线，调整计算资源。这种方法无法适应弹性业务需求，对于实时响应突发流量的能力有限。同时，无法追踪到历史数据中的趋势性变化，可能导致资源分配不合理。
4. **多指标阈值收缩**：在这种方法中，根据多个指标和预设阈值调整计算资源。尽管这种方法相对于其他方法更为精确，但其效果取决于指标选取和阈值设定。若未充分考虑这些因素，可能会影响系统性能，导致资源分配不合理。并且依旧无法考虑到业务发展的趋势性。

可观察到，现有的自动伸缩方法均存在不足，无法在不同场景下完美适应需求。因此，需要进一步研究和发展更为智能、灵活的自动伸缩策略，以满足多样化的计算资源需求。

#### 2.3 工作量预测方法

为了保证 Serverless 应用程序在业务中的可靠性，应当设计一种有效的自动伸缩算法。为此，我们采用了基于工作量的 Serverless 计算自动伸缩算法，结合负载预测和自动伸缩策略。在本节中，我们将详细介绍三种工作量预测方法：移动平均（MA）、向量自回归（VAR）以及差分整合移动平均自回归（ARIMA）模型。

##### **2.3.1 MA**（移动平均，作为基线方法）

移动平均（Moving Average，简称 MA）是一种简单的时间序列预测方法，它通过计算一定时间窗口内的平均值来预测未来的数据。在 Serverless 计算的负载预测中，可以采

用 MA 方法来预测未来一段时间内的工作量。给定一个时间序列数据集 {*x*1*,x*2*,...,xn*}，移动平均法用长度为 *m* 的滑动窗口来计算序列的平均值，计算公式如下：

##### 式（2-1）

MA 方法简单易实现，但其预测精度受到时间窗口长度的影响，较长的时间窗口可能导致预测滞后，较短的时间窗口可能导致预测不稳定。

我们选择移动平均作为基准线方法的原因在于，首先它简单易实现：相较于其他复杂的预测方法，移动平均法的实现过程非常简单。只需要计算一定时间窗口内的数据平均值，无需涉及复杂数学模型。这使得移动平均方法易于理解和快速实现。其次，它拥有低计算成本。移动平均法的计算复杂度较低，不需要进行复杂的参数估计和优化。这意味着移动平均法在计算资源有限的场景下，仍然能够保持较好的预测性能。再者，移动平均法具有很好的平滑效果，能够有效消除短期的波动和噪声，使预测结果更加稳定。这对于某些具有高噪声或波动性的场景（如负载预测）来说，是非常有价值的。

###### **2.3.2 VAR**（向量自回归）

向量自回归（VAR）模型是一种用于多变量时间序列数据的预测方法。VAR 模型通过将每个时间序列作为其他时间序列的滞后值（lagged values）的线性函数来进行建模。

VAR 模型具有预测多元时间序列变量之间关系的优势，可以捕捉变量间的动态相互影响。

##### 假设我们有 *n* 个时间序列数据 *yt* = (*y*1*t,y*2*t,*··· *,ynt*)，则 *p* 阶 VAR 模型可表示为：

###### *yt* = *c* + *A*1*yt*−1 + *A*2*yt*−2 + ··· + *Apyt*−*p* + *et* 式（2-2）

其中，*t* = 1*,*2*,*··· *,T*，*c*是一个*n*维常数向量，*Ai* 是一个*n*×*n*维系数矩阵，*et* 是一个 *n* 维误差向量，满足 *E*(*et*) = 0 且协方差矩阵为常数矩阵，即 *E*(*ete*′*t*) = Σ，*E*(*ete*′*t*−*j*) = 0 对于所有的 *j* = 0̸ 。

举个例子，为了预测 Serverless 计算的工作量，我们首先需要收集多元时间序列数据，例如请求数、响应时间和 CPU 使用率等。接着可以使用 VAR 模型来预测未来的工作量。

假设我们有三个时间序列数据：请求数 *xt*、响应时间 *yt* 和 CPU 使用率 *zt*，可以构建一个二阶 VAR 模型如下：

[

*xt yt zt*] = *c* + *A*1 [*xt*−1 *yt*−1 *zt*−1] + *A*2 [*xt*−2 *yt*−2 *zt*−2] + *et* 式（2-3）

##### 通过拟合此 VAR 模型，可以预测未来的工作量，从而为 Serverless 计算提供合适的

自动伸缩策略。

###### **2.3.3 ARIMA**（自回归差分移动平均）

ARIMA（Autoregressive Integrated Moving Average，自回归差分移动平均）是一种广泛应用于时间序列预测的方法。ARIMA 模型由三个主要部分组成：自回归（AR），差分（I）和移动平均（MA）。

给定一个时间序列数据 *yt*，ARIMA 模型表示为[3]：

Φ(*B*)(1 − *B*)*dyt* = Θ(*B*)*et* 式（2-4）

其中，*B* 是后向移位算子（即 *Byt* = *yt* 1），*d* 是差分阶数，Φ(*B*) 和 Θ(*B*) 分别是 *p*

阶自回归多项式和 *q* 阶

Φ(*B*) = 1 − *ϕ*1*B* − *ϕ*2*B*2 − ··· − *ϕpBp*

式（2-5）

##### Θ(*B*) = 1 − *θ*1*B* − *θ*2*B*2 − ··· − *θqBq* 式（2-6）

可以将 ARIMA 模型表示为 ARIMA(*p,d,q*)，其中 *p* 是自回归项的阶数，*d* 是差分的阶数，*q* 是移动平均项的阶数。

为了预测 Serverless 计算的工作量，我们首先需要选择合适的 *p*、*d* 和 *q* 参数。可以通过观察时间序列的自相关图（ACF）和偏自相关图（PACF）来确定这些参[4] ，或者使用自动参数选择方法，如 Akaike Informatio n Criterion (AIC) 或 Bayesian Information

Criterion (BIC)。

###### 例如，假设我们选择了 ARIMA(2*,*1*,*1) 模型来预测请求数量 *xt*，模型可以表示为：

(1 − *ϕ*1*B* − *ϕ*2*B*2)(1 − *B*)*xt* = (1 − *θ*1*B*)*et* 式（2-7）

通过拟合此 ARIMA 模型，可以预测未来的工作量，从而为 Serverless 计算提供合适的自动伸缩策略。

为了提高预测的准确性，可以结合 MA、VAR 和 ARIMA 模型进行预测和性能对比，根据实际情况选择合适的方法。

2.3.4 Prophet

2.3.4.1 简介

Prophet 是由 Facebook 的核心数据科学团队开发的开源算法，专门用于进行时间序列预测。它设计的目标是使得时间序列预测对于业务问题更加适用，包括那些存在一些实际的、复杂的趋势的问题，如周期性变化、季节性效应和假日效应等。[5]

2.3.4.2 模型原理

Prophet 模型基于分解的时间序列预测方法，主要包括三个成分：趋势 (Trend)、季节性 (Seasonality) 和假日效应 (Holidays)。趋势部分旨在捕获数据中的长期趋势，季节性部分则用来捕获周期性模式，而假日效应则针对不规则的事件，例如公众假日或其他特殊事件。

这种分解方法基于以下的加法模型：

*y*(*t*) = *g*(*t*) + *s*(*t*) + *h*(*t*) + *ϵt* 式（2-8）

其中，*y*(*t*) 是观测值，*g*(*t*) 是趋势成分，*s*(*t*) 是季节性成分，*h*(*t*) 是假日效应，*ϵt* 是误差项。

**趋势成分** *g*(*t*)**：** Prophet 模型假设趋势成分可以通过非线性函数进行适应，而这个非线性函数的形状由数据的变化点决定。在这种设置下，趋势可以在变化点灵活地改变其增长速率。具体来说，对于没有饱和上限和下限的趋势，Prophet 使用以下的分段线性模型：

###### 式（2-9）

其中，*S* 是变化点的数量，*τi* 是第 *i* 个变化点的位置，*ai* 和 *bi* 是在第 *i* 个变化点的增长率和偏移量的变化。*k* 和 *m* 是初始化的增长率和偏移量。

**季节性成分** *s*(*t*)**：** Prophet 模型采用傅里叶级数来适应季节性效应，可以捕获不同周期的季节性变化。具体来说，年度和周度季节性模式可以通过以下傅里叶级数进行建模：

###### 式（2-10）

其中，*P* 是季节性周期，*N* 是傅里叶级数的阶数，*an* 和 *bn* 是要估计的参数。

**假日效应** *h*(*t*)**：**

在许多实际应用中，特定的日期或事件（如公众假期，特定的营销活动或其他特殊事件）可能会对时间序列产生显著影响。这些影响可以被视为” 假日效应”，它们会在特定日期或事件期间突然改变时间序列的行为。这种假日效应可以通过添加额外的回归项来建模。具体来说，对于每一个假日，都会创建一个指示函数，该函数在假日期间为 1，其他时间为 0。然后，这个指示函数与一个回归系数相乘，该系数表示假日的影响强度。所有的假日效应可以表示为：

###### 式（2-11）

其中，*H* 是假日的数量，*Di* 是第 *i* 个假日的日期，*ki* 是对应的回归系数，1(·) 是指示函数。

假日效应的参数 *ki* 是通过模型拟合过程中的数据来估计的。由于假日效应可能在不同的假日之间有所不同，每个假日都有自己的回归系数。

在实际应用中，用户需要提供一个假日的列表，并指定每个假日的日期。此外，用户还可以为每个假日设置一个“窗口”，表示假日效应可以影响的日期范围。

2.3.4.3 贝叶斯推断和参数估计

Prophet 模型对模型参数的估计是基于贝叶斯推断的。这意味着我们不仅仅是找到一个最优的参数值，而是要找到参数的概率分布，这样我们就可以考虑到参数估计的不确定性。

贝叶斯推断的基本思想是结合先验知识和观测数据来估计模型参数。对于每一个参数，应当指定一个先验分布，这个分布表示了在观测数据之前我们对参数的信念。然后，我们使用观测数据来更新这个信念，得到参数的后验分布。

在 Prophet 模型中，我们对每一个参数 *θ*（可以是趋势、季节性或假日效应的参数）的先验分布 *p*(*θ*) 做出某些假设。然后，我们使用观测数据 *y* 来计算参数的似然函数 *L*(*θ*|*y*)，这个函数表示在给定参数 *θ* 的情况下，观测数据 *y* 的概率。最后，我们使用贝叶斯定理来计算参数的后验分布：

###### 式（2-12）

在这个公式中，*p*(*y*) 是观测数据的概率，我们通常不直接计算这个值，而是通过对参数的所有可能值的后验分布进行归一化来得到。

在实际的计算中，由于后验分布可能是复杂的，我们通常使用马尔科夫链蒙特卡洛

（MCMC）方法[6] 或者变分贝叶斯方法[7] 来近似计算后验分布。

此处，参数的先验分布通常是正态分布或者拉普拉斯分布，这取决于我们对参数的先验知识。例如，如果我们认为一个参数应该接近 0，那么可以选择均值为 0 的正态分布作为先验分布。

## 第三章 算法的设计与实现

算法的设计和实现，需要基于对时间序列数据的特性和规律的分析和建模。这些特性包括趋势、季节性、周期性、自相关性等。基于这些特性，可以设计出各种不同的时间序列预测算法。前文已经介绍了本研究将采用的预测算法，在实际使用时间序列预测算法进行预测之前，原始的数据可能存在缺失值和异常值、数据的格式不规范等问题。

应当对数据进行预处理。在此之后基于标准化后的数据应用具体的算法。整体设计思路如图 3-1 所示。

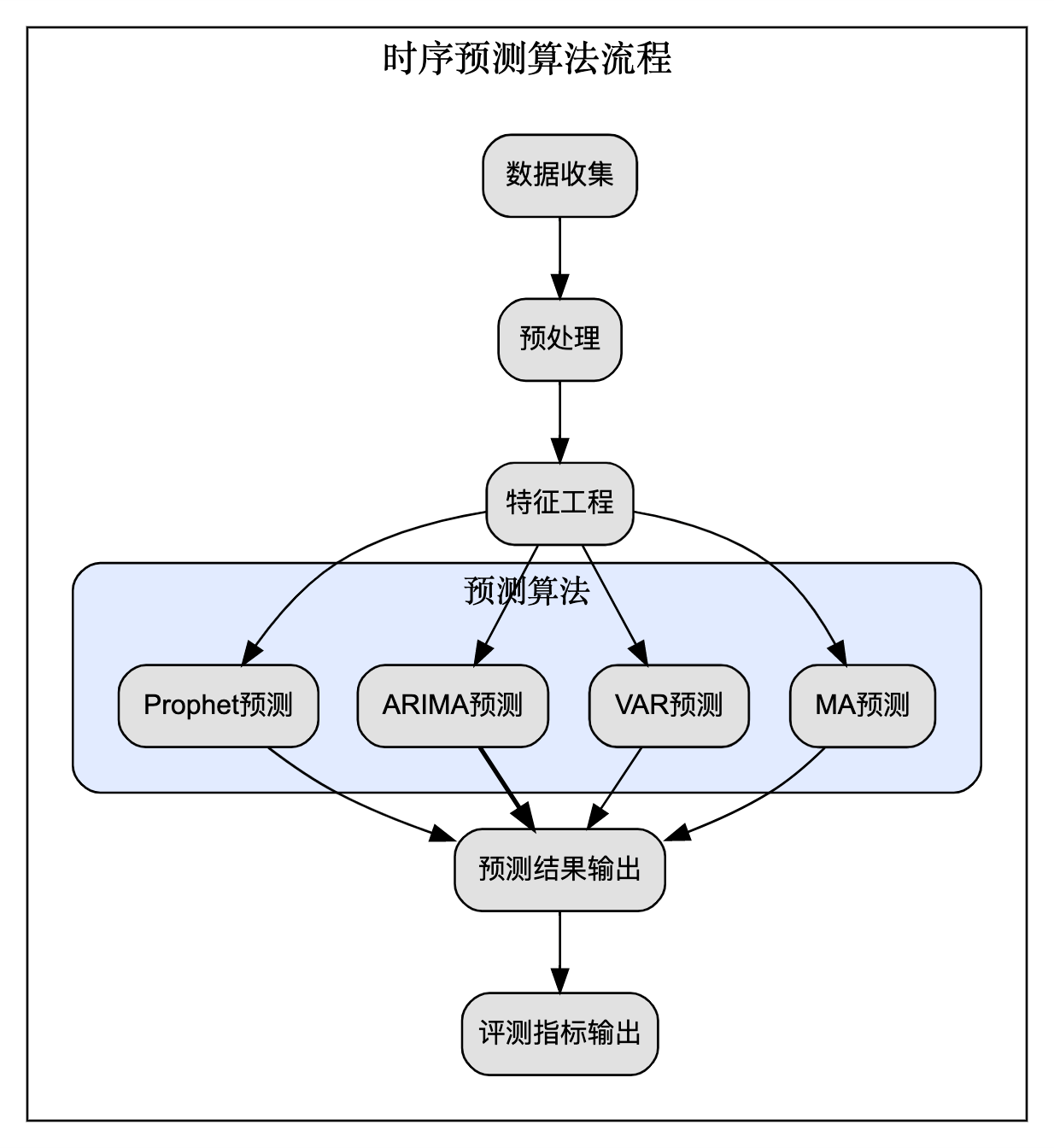


图 3-1 时序算法预测流程

### **3.1** 准备工作

#### 3.1.1 数据收集与预处理

数据收集和预处理是时间序列预测任务的关键步骤，可以确保我们拥有无缺失值、无异常值、可靠和有代表性的数据，以便进行准确的预测。在本研究中，我们使用 Binance 数字货币交易所交易 API 调用历史数据公开数据集作为数据来源。

我们从 Binance 提供的公开数据集中获取交易 API 调用历史数据。原始数据以 zip 格式存储，每日一个文件。对于每个文件，解压后得到一个 csv 文件，在这个 csv 文件中，每行代表一次交易调用记录。数据包含多个字段，其中 time 字段表示调用发生的时间，单位为毫秒时间戳。其余字段本研究不涉及。

为便于分析和预测，应当对原始数据进行预处理。首先，将毫秒时间戳转换为秒时间戳，以便于后续处理。接着，对数据进行聚合处理，将每个小时内的调用次数汇总，生成一个新的数据集。新数据集包含两列：time 和 event\_count。其中，time 代表小时所在的秒时间戳，event\_count 代表该小时时间段内的调用次数。

预处理的具体步骤如下：

1. 加载原始数据集。
2. 按小时对数据进行分组聚合，计算每个小时内的调用次数。
3. 将 time 字段的毫秒时间戳转换为秒时间戳。
4. 生成新的数据文件，每个文件是一天的调用统计，包含 time 和 event\_count 两列。
5. 将每个月的所有文件聚合为一个文件

经过上述预处理步骤，我们得到了一个无缺失值、无异常值、有序的数据集。

#### 3.1.2 特征工程

特征工程是机器学习和预测任务中的重要环节，可以提取有意义的信息以提高预测性能。由于调用频次与时间通常有较强的相关性，我们从时间戳中提取 weekday 和 hour 两列特征。

图 3-2 展示不同时间点的调用次数分布，可直观观察到数据并非噪声或随机游走，而是存在一定的规律性，因此存在被预测的可能。图 3-3 展示一天中不同小时的调用次数分布，可直观观察到存在峰值时段和谷值时段。图 3-4 a展示一周中不同天（周一至周日）的调用次数分布。图 3-4 b展示 time、event\_count、weekday 和 hour 之间的相关性矩阵。

通过分析上述图表和相关性矩阵，可以发现调用次数与 weekday 和 hour 特征具有一定的相关性。这些特征将有助于我们建立更准确的工作量预测模型。

##### **3.1.3** 模型选择与评估

在本文中，我们采用了 MA、VAR、ARIMA、Prophet 四种时间序列模型来进行负载预测。相较于基于神经网络的机器学习模型，我们选择此类统计学模型的原因主要有以下方面：

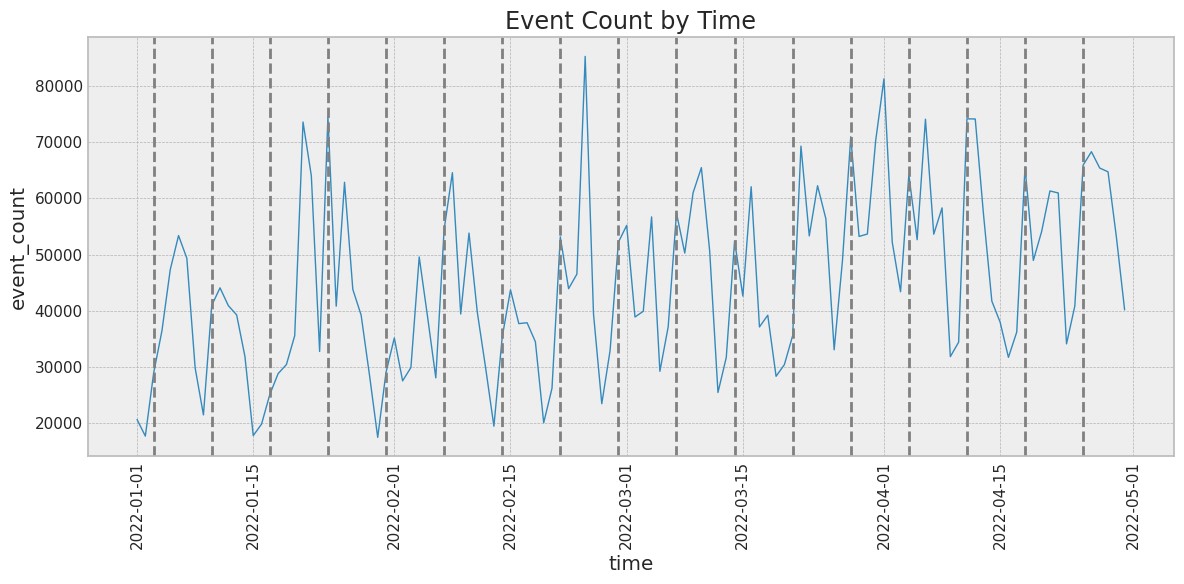


图 3-2 事件计数随时间变化图

表 3-1 自相关和偏自相关在不同滞后阶数的值

|  |  |  |
| --- | --- | --- |
| **滞后阶数（Lags）** | **自相关性（Autocorrelation）** | **偏自相关性（PACF）** |
| 0 | 1.00 | 1.00 |
| 1 | 0.60 | 0.60 |
| 2 | 0.44 | 0.13 |
| 3 | 0.34 | 0.06 |
| 4 | 0.29 | 0.06 |
| 5 | 0.27 | 0.07 |
| 6 | 0.24 | 0.03 |
| 7 | 0.19 | -0.02 |
| 8 | 0.16 | 0.01 |
| 9 | 0.16 | 0.04 |
| 10 | 0.18 | 0.06 |
| 11 | 0.16 | -0.00 |
| 12 | 0.16 | 0.04 |
| 13 | 0.16 | 0.03 |
| 14 | 0.12 | -0.03 |
| 15 | 0.11 | -0.00 |
| 16 | 0.11 | 0.03 |
| 17 | 0.13 | 0.04 |
| 18 | 0.17 | 0.07 |
| 19 | 0.18 | 0.04 |
| 20 | 0.17 | 0.01 |
| 21 | 0.18 | 0.04 |
| 22 | 0.21 | 0.08 |
| 23 | 0.24 | 0.07 |
| 24 | 0.27 | 0.07 |

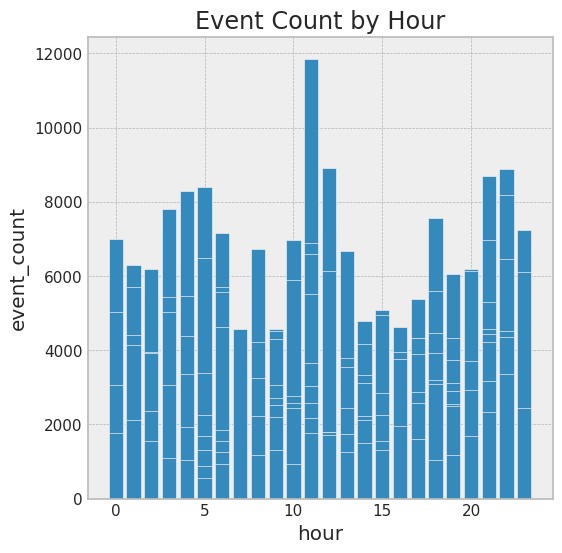


图 3-3 事件计数分布（按小时）

(a) 事件计数分布图（按weekday） (b) 相关性矩阵图

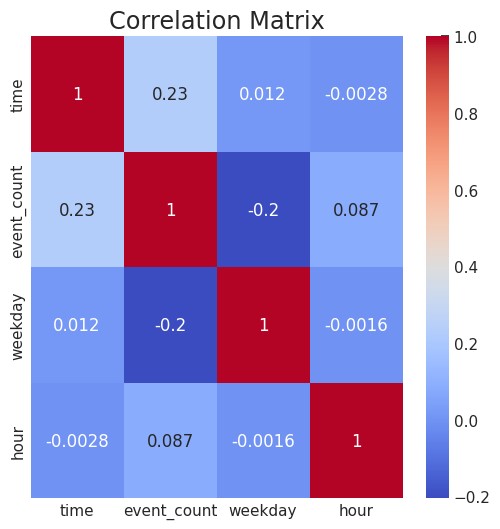
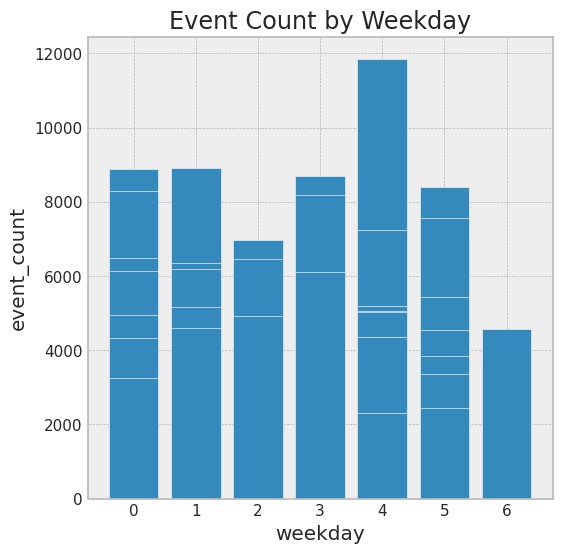


图 3-4 weekday 分布和相关性矩阵

第一，时间序列模型适用于具有时间依赖性的数据，可以更好地捕捉数据的周期性、趋势性和季节性等特征。在 Serverless 计算中，负载数据通常具有时间相关性，因此时间序列模型是一种自然的选择。

第二，时间序列模型具有可解释性和可靠性，可以更好地理解模型的运作原理，评估模型的准确性和稳定性。对于负载预测来说，准确性和稳定性是至关重要的，因此时间序列模型可以为我们提供更可靠的负载预测结果。

最后，非常重要的一点是，时间序列模型相较于机器学习模型，需要的数据量更小，模型训练和预测速度更快。在 Serverless 计算中，资源的快速响应是至关重要的，而历史数据量往往只有几个星期，使用机器学习模型难以较好收敛，因此时间序列模型可以更快地生成负载预测结果，以满足 Serverless 应用程序对计算资源的实时需求。

#### 本研究选择的各模型各有特点。MA（Moving Average）模型主要用于捕捉负载数据

的短期波动性，本研究选择其作为基准（Baseline）模型。VAR（Vector Autoregression）模型则可以同时考虑多个变量之间的相互作用，从而验证时间中提取的特征是否能改善预测效果，ARIMA （Autoregressive Integrated Moving Average）模型则是一种广泛应用的时间序列模型，可以同时考虑趋势、周期和季节性等因素，并且能够自动从数据中学习到这些因素，相比 VAR 在保证训练效率的前提下，更加智能。Prophet 则是一个更加新颖的模型，能够考虑到节假日等因素的影响。通过多种模型的组合和比较，从而在实际应用选择最适合负载预测的模型。

表 3-2 MA 模型评测指标

|  |  |
| --- | --- |
| **指标** | **数值** |
| MSE | 1139965.31 |
| RMSE | 1067.69 |
| MAE | 805.27 |
| MAPE | 39.11% |
| SMAPE | 34.47% |

### **3.2** 自动伸缩算法设计与实现

#### 3.2.1 MA 模型

MA（Moving Average），即移动平均模型是一种常用的时间序列模型，用于捕捉负载数据的短期波动性。在本文中，本研究将 MA 模型作为基准模型（Baseline Model）来进行负载预测，以评估其他模型的优劣性。

具体地，本研究采用了 WIN\_SIZE=3 的 MA 模型来进行负载预测。将历史数据按照 WIN\_SIZE 划分成若干个时间窗口，每个时间窗口内的数据点的平均值被用来作为该时间窗口的预测值。通过这样的方式，可以捕捉负载数据的短期波动性，同时减少数据的随机波动性。

为了评估 MA 模型（以及后续其他模型）的预测性能，本研究使用均方误差（Mean Squared Error）、均方根误差（Root Mean Squared Error）、平均绝对误差（Mean Absolute Error）、平均绝对百分比误差（Mean Absolute Percentage Error）和对称平均绝对百分比误差（Symmetric Mean Absolute Percentage Error）等指标来评估模型的预测准确性和稳定性。在 WIN\_SIZE=3 的情况下，我们得到了 3-2 所示模型预测性能。

图 3-6 展示 MA 模型的预测结果和实际负载值的对比图。从图中可以看出，MA 模型能够较为准确地捕捉到负载数据的短期波动性，预测结果与实际负载值的变化趋势基本一致，但预测效果不是很理想。同时，此图也包含 MA 模型的残差图，可观察到 MA 模型的残差依旧存在一些周期性，这表明存在改进的空间。

MA 模型可以为我们提供一个基础的负载预测结果，同时也可以用来评估其他更复杂的负载预测模型的性能优劣。作为基准模型，本研究不期望其实际应用效果，原因在于 MA 模型虽然简单，但在某些情况下可能不适用。例如，在负载数据存在长期趋势或季节性变化的情况下，MA 模型可能无法捕捉到这些特征，从而导致负载预测的准确性下降。

#### 3.2.2 VAR 模型

VAR 模型是一种常用的多变量时间序列模型，可以同时考虑多个变量之间的相互作用，适用于具有相互关联的多个时间序列数据的预测。在本文中，我们选用 time、

weekday 和 hour、event\_count 等多个特征来构建 VAR 模型。

#### 为了解决模型选择问题，评估模型的好坏，并寻找最佳的模型我们使用一些准则，

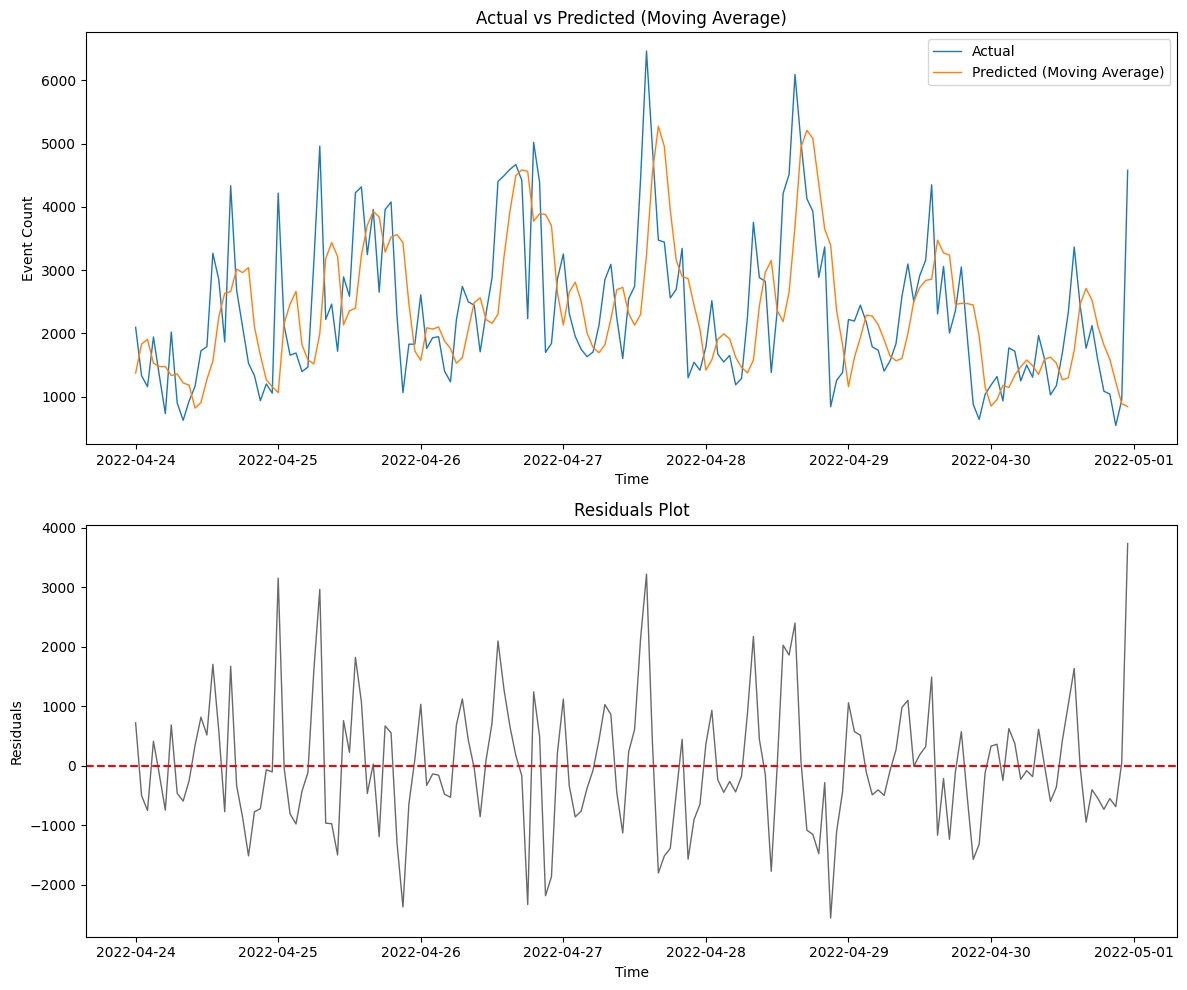


图 3-5 MA 预测结果和实际负载值、残差对比表 3-3 VAR 不同阶数下的各准则值

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VAR Order** | **AIC** | **BIC** | **FPE** | **HQIC** |
| 0\* | 19.34 | 19.35 | 2.518e+08 | 19.35 |
| 1 | 15.22 | 15.25 | 4.090e+06 | 15.23 |
| 2 | 15.18 | 15.23 | 3.907e+06 | 15.20 |
| 3 | 15.17 | 15.23 | 3.858e+06 | 15.19 |
| 4 | 15.16 | 15.25 | 3.823e+06 | 15.19 |
| 5 | 15.15 | 15.26 | 3.781e+06 | 15.19 |
| 6 | 15.14 | 15.27 | 3.748e+06 | 15.18 |
| 7 | 15.13 | 15.28 | 3.725e+06 | 15.19 |
| 8 | 15.12 | 15.29 | 3.693e+06 | 15.18 |
| 9 | 15.10 | 15.29 | 3.615e+06 | 15.17 |
| 10 | 15.08 | 15.29 | 3.546e+06 | 15.16 |
| 11 | 15.07 | 15.30 | 3.497e+06 | 15.15 |
| 12 | 15.05 | 15.30 | 3.421e+06 | 15.14 |
| 13 | 15.02 | 15.30 | 3.342e+06 | 15.12 |
| 14 | 14.99 | 15.29 | 3.251e+06 | 15.10 |
| 15 | 14.96 | 15.28 | 3.144e+06 | 15.08 |
| 16 | 14.91 | 15.25 | 2.994e+06 | 15.03 |
| 17 | 14.85 | 15.21 | 2.826e+06 | 14.98 |
| 18 | 14.77 | 15.15 | 2.599e+06 | 14.91 |
| 19 | 14.66 | 15.06 | 2.322e+06 | 14.80 |
| 20 | 14.48 | 14.90 | 1.942e+06 | 14.63 |
| 21 | 14.20 | 14.64 | 1.473e+06 | 14.36 |
| 22 | 13.62 | 14.08 | 8.196e+05 | 13.78 |
| 23 | -41.31 | -40.82 | 1.152e-18 | -41.13 |
| 24 | -46.04 | -45.53 | 1.016e-20 | -45.85 |

包括赤池信息准则（Akaike Information Criterion, AIC）、贝叶斯信息准则（Bayesian Information Criterion, BIC）、预测误差准则（Final Prediction Error, FPE）以及汉纳-奎因准则（Hannan-Quinn Information Criterion, HQIC）。

**赤池信息准则（Akaike Information Criterion, AIC）** ：AIC 是由 Akaike 提出的[8]，主要用于模型选择。AIC 不仅考虑了模型的拟合度，还对模型的复杂度进行了惩罚。当模型越复杂（参数越多）时，AIC 值就越大。选择最佳模型时，通常选择 AIC 值最小的模型。AIC 的计算公式如下：

#### *AIC* = 2*k* − 2ln(*L*)

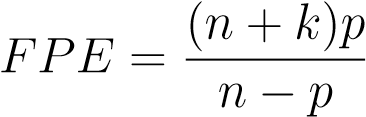
其中，*k* 是模型参数的数量，*L* 是模型的最大似然值。

**贝叶斯信息准则（Bayesian Information Criterion, BIC）** ：BIC 也是用于模型选择的一种准则，由 Schwarz 提出[9]。与 AIC 类似，BIC 也对模型的复杂度进行了惩罚，但是 BIC 的惩罚项比 AIC 更大，因此 BIC 对模型的复杂度惩罚更严重。BIC 的计算公式如下：

#### *BIC* = ln(*n*)*k* − 2ln(*L*)

其中，*n* 是观察的数据量，*k* 是模型参数的数量，*L* 是模型的最大似然值。

**预测误差准则（Final Prediction Error, FPE）** ：FPE 由 Akaike 提出，主要用于时间序列模型的选择。FPE 不仅考虑了模型的拟合度，还对模型的复杂度进行了惩罚[10]。FPE 的计算公式如下：



其中，*n* 是观察的数据量，*p* 是模型参数的数量。

#### 汉纳-奎因准则（Hannan-Quinn Information Criterion, HQIC） ：HQIC 由汉纳和奎因

提出，主要用于模型选择。与 AIC 和 BIC 类似，HQIC 也对模型的复杂度进行了惩罚，但是 HQIC 的惩罚项介于 AIC 和 BIC 之间[11]。HQIC 的计算公式如下：

##### *HQIC* = −2ln(*L*) + 2*k* ln(ln(*n*))

其中，*n* 是观察的数据量，*k* 是模型参数的数量，*L* 是模型的最大似然值。

表 3-3 展示不同阶数的 VAR 模型的 AIC、BIC、FPE 和 HQIC 值。从表中可观察到，当阶数为 22 时，VAR 模型的 AIC、BIC、FPE 和 HQIC 值均达到最小，并且没有变成负值，因此我们选择阶数为 22 的 VAR 模型。

表 3-4 展示使用 VAR 模型进行负载预测的评价指标。

表 3-4 VAR 模型评测指标

|  |  |  |
| --- | --- | --- |
| **指标** | **数值** | **MA 模型基准** |
| MSE | 680996.57 | 1139965.31 |
| RMSE | 825.23 | 1067.69 |
| MAE | 590.93 | 805.27 |
| MAPE | 26.98% | 39.11% |
| SMAPE | 25.54% | 34.47% |

实验结果表明，使用 VAR 模型进行负载预测的效果明显优于 MA 模型。

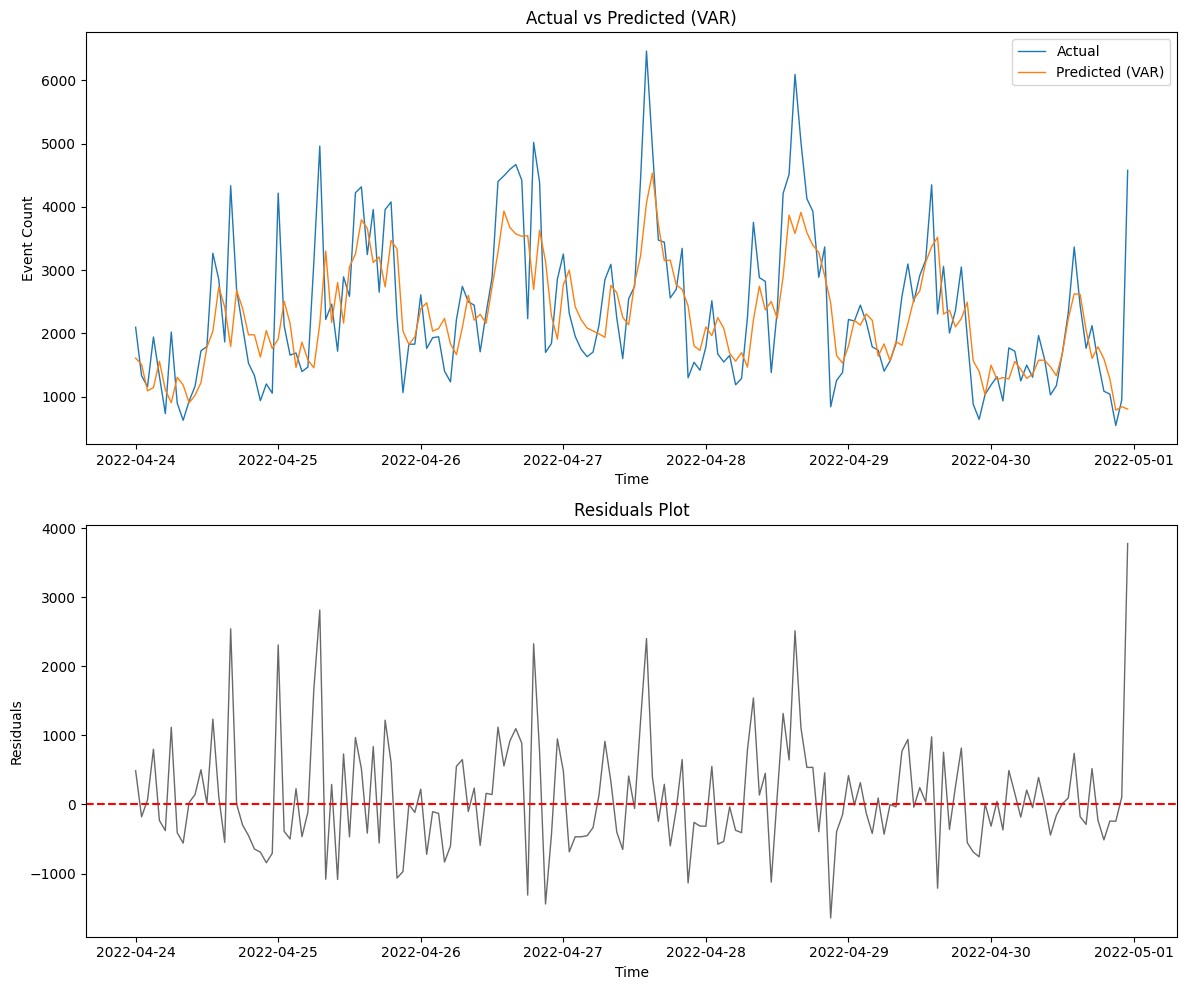


图 3-6 预测结果和实际负载值、残差对比

图 3-6 展示使用 VAR 模型进行负载预测的实验结果。可观察到预测值与实际值的误差较小，且残差图表明基本消除了周期性，且预测误差更加均匀，这说明 VAR 模型在捕捉负载数据的周期性、趋势性和季节性等方面具有较好的能力。

###### **3.2.3 ARIMA** 模型

3.2.3.1 模型参数选择

在时间序列分析中，ARIMA（Autoregressive Integrated Moving Average）模型是一种广泛应用的时间序列预测方法。ARIMA 模型的参数选择是关键步骤之一，本节将介绍基于信息准则的模型参数选择方法。

首先，应当对时间序列数据进行平稳性检验，若序列不满足平稳性，则需要进行差分处理，直至达到平稳状态。通过对差分后的序列进行自相关函数（ACF）和偏自相关函数（PACF）的分析，可以初步确定 ARIMA 模型的参数 p 和 q。其中，p 表示 AR 模型的阶数，q 表示 MA 模型的阶数。

在本研究中，ARIMA 模型的参数选择过程采用了 Akaike Information Criterion (AIC) 和 Bayesian Information Criterion (BIC) 两个指标。AIC 和 BIC 均可以用于衡量模型拟合优度，它们分别对应模型拟合优度与模型复杂度的平衡，较小的 AIC 或 BIC 值表示更好的模型拟合。

首先，我们根据给定数据，计算后生成自相关函数（ACF）和偏自相关函数（PACF）图表，如表 3-5 所示。

然后，我们根据 ACF 和 PACF 图形（如图 3-7 所示）确定 ARIMA 模型的参数 p 和 q。从图中可观察到，ACF 图在滞后阶数为 1 时截尾，PACF 图在滞后阶数为 2 时截尾，因此可以初步确定 ARIMA 模型的参数 p=1，q=2。

为了进一步确定 ARIMA 模型的参数，选取不同的 ARIMA 模型参数进行尝试。采用的是穷举并对比 AIC 和 BIC 的方法。

表 3-6 展示针对给定时间序列数据的不同 ARIMA 模型的 AIC 和 BIC 值。

经过比较，我们发现 ARIMA(2, 0, 2) 模型具有最低的 AIC 值为 47672.61，同时 BIC 值为 47708.40，因此选取 ARIMA(2, 0, 2) 作为最优模型。

3.2.3.2 ARIMA 模型的预测效果评估

图 3-8 展示 ARIMA 模型的预测结果和实际负载值的对比图。从图中可以看出，

ARIMA 模型相比 MA 和 VAR 模型，能够更为准确地捕捉到负载数据的短期波动性，预测结果与实际负载值的变化趋势基本一致，预测效果较好。同时，图中还包含 ARIMA 模型的残差图，可观察到周期性基本被消除。

从表 3-7 中可观察到，相较于 MA 和 VAR 模型，ARIMA 模型的预测效果评估指标表现良好。均方误差和均方根误差较小，说明模型的预测误差较小；平均绝对误差和平均绝对百分比误差也较小，说明模型的平均预测误差不大；对称平均绝对百分比误差也较小，说明模型对正负误差的惩罚是比较对称的。

表 3-5 ACF 和 PACF 表

|  |  |  |
| --- | --- | --- |
| lags | autocorrelation | pacf |
| 0 | 1.00 | 1.00 |
| 1 | 0.60 | 0.60 |
| 2 | 0.44 | 0.13 |
| 3 | 0.34 | 0.06 |
| 4 | 0.29 | 0.06 |
| 5 | 0.27 | 0.07 |
| 6 | 0.24 | 0.03 |
| 7 | 0.19 | -0.02 |
| 8 | 0.16 | 0.01 |
| 9 | 0.16 | 0.04 |
| 10 | 0.18 | 0.06 |
| 11 | 0.16 | -0.00 |
| 12 | 0.16 | 0.04 |
| 13 | 0.16 | 0.03 |
| 14 | 0.12 | -0.03 |
| 15 | 0.11 | -0.00 |
| 16 | 0.11 | 0.03 |
| 17 | 0.13 | 0.04 |
| 18 | 0.17 | 0.07 |
| 19 | 0.18 | 0.04 |
| 20 | 0.17 | 0.01 |
| 21 | 0.18 | 0.04 |
| 22 | 0.21 | 0.08 |
| 23 | 0.24 | 0.07 |
| 24 | 0.27 | 0.07 |

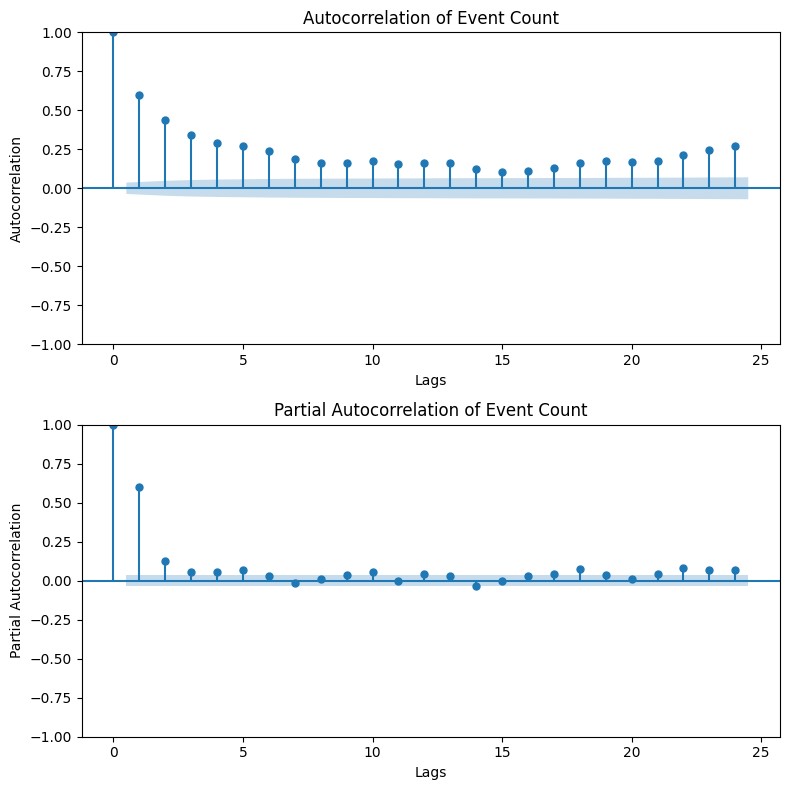


图 3-7 ACF 和 PACF 图形表 3-6 ARIMA 模型评测指标

|  |  |  |
| --- | --- | --- |
| **ARIMA Parameters** | **AIC** | **BIC** |
| (0, 0, 0) | 49040.07 | 49052.00 |
| (0, 0, 1) | 48195.20 | 48213.10 |
| (0, 0, 2) | 47930.20 | 47954.06 |
| (0, 0, 3) | 47843.45 | 47873.28 |
| (0, 0, 4) | 47811.97 | 47847.77 |
| (0, 0, 5) | 47786.11 | 47827.87 |
| (0, 0, 6) | 47739.97 | 47787.69 |
| (0, 0, 7) | 47716.49 | 47770.18 |
| (1, 0, 0) | 47764.59 | 47782.48 |
| (1, 0, 1) | 47707.93 | 47731.79 |
| (1, 0, 2) | 47697.82 | 47727.65 |
| (1, 0, 3) | 47687.50 | 47723.29 |
| (1, 0, 4) | 47680.53 | 47722.29 |
| (1, 0, 5) | 47681.68 | 47729.40 |
| (1, 0, 6) | 47681.96 | 47735.65 |
| (1, 0, 7) | 47671.27 | 47730.93 |
| (2, 0, 0) | 47718.77 | 47742.63 |
| (2, 0, 1) | 47679.91 | 47709.74 |
| **(2, 0, 2)** | **47672.61** | **47708.40** |
| (2, 0, 3) | 47674.64 | 47716.40 |
| (2, 0, 4) | 47675.70 | 47723.42 |
| (2, 0, 5) | 47683.87 | 47737.56 |
| (2, 0, 6) | 47685.69 | 47745.35 |
| (2, 0, 7) | 47671.74 | 47737.36 |

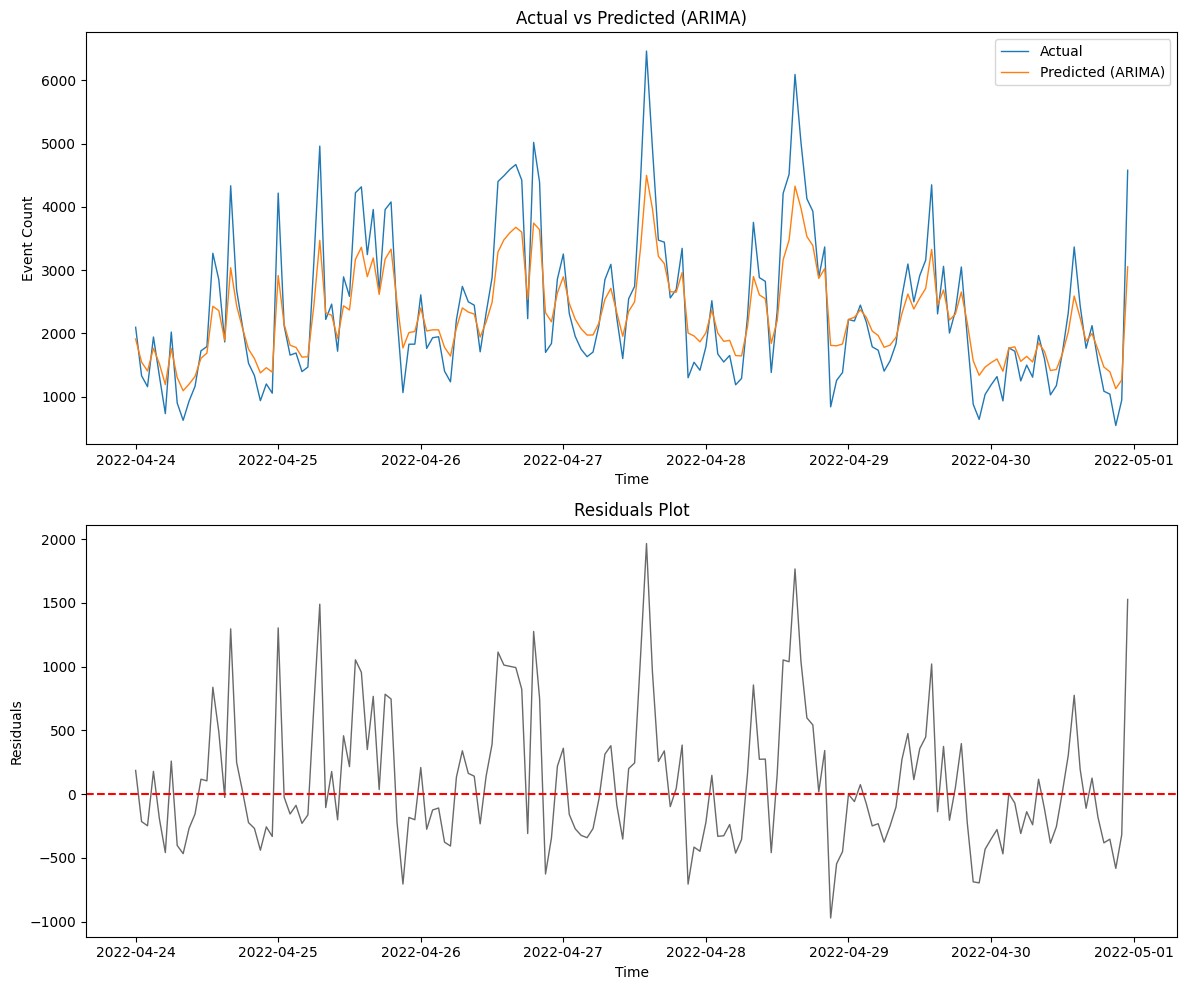


图 3-8 预测结果和实际负载值、残差对比表 3-7 ARIMA 模型预测效果指标

|  |  |  |
| --- | --- | --- |
| **指标** | **数值** | **MA 模型基准** |
| MSE | 291700.66 | 1139965.31 |
| RMSE | 540.09 | 1067.69 |
| MAE | 405.24 | 805.27 |
| MAPE | 19.30% | 39.11% |
| SMAPE | 17.82% | 34.47% |

###### **3.2.4 Prophet** 模型

3.2.4.1 数据预处理

由于 Prophet 模型所需要的数据格式特殊，首先需要对数据再次预处理。

Prophet 模型要求时间序列的时间戳是 ds（日期戳）列，并且它必须是 datetime 类型。在本例中，时间戳数据被存储在 time 列中，并且以秒为单位。故使用 pd.to\_datetime 函数将其转换为 datetime 类型。

然后，Prophet 模型要求时间序列的值被存储在 y 列中。在本研究的数据中，这些值被存储在 event\_count 列中，因此应当将列名更改为 y。

最后，应当将数据集划分为训练集和测试集。在这个例子中，我们选择了 2022 年 1 月 1 日到 2022 年 4 月 30 日的数据，并且将最后一周的数据作为测试集，其余的数据作为训练集。

3.2.4.2 模型训练

在完成数据预处理后，可以开始训练 Prophet 模型。模型训练主要包括模型的初始化和拟合。

首先，应当创建一个 Prophet 模型的实例。在默认情况下 Prophet 模型会自动处理趋势和季节性，但是也可以通过设置参数来调整模型的行为。例如，可以通过设置 seasonality\_mode 参数来改变季节性的模型（加性或乘性），或者通过设置 changepoint\_prior\_scale 参数来改变趋势变化点的灵敏度。在本例中，我们使用默认设置创建一个 Prophet 模型的实例。

3.2.4.3 Prophet 模型的预测效果评估

图 3-9 展示 Prophet 模型的预测效果。蓝色曲线代表实际值，黄色曲线代表预测值的期望。我们还可以看到一个灰色区域，灰色区域的上边界代表预测值的上界，下边界代表预测值的下界。

图 3-10 展示 Prophet 模型的预测残差图。

我们对 Prophet 模型的预测性能进行了定量评估，所得结果如表 3-8 所示。结果表明，Prophet 模型在所有指标上均优于 MA 模型基准。

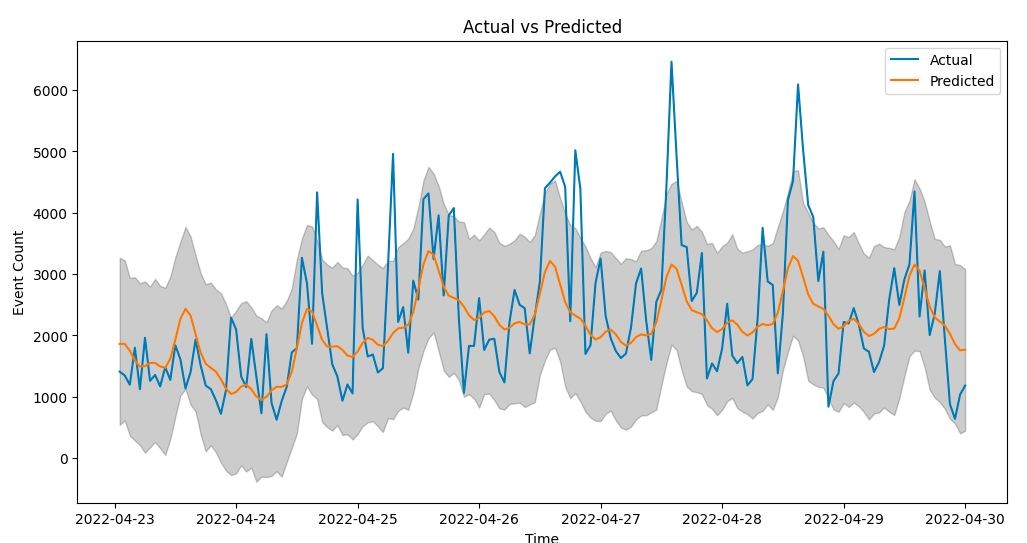


图 3-9 预测结果和实际负载值对比

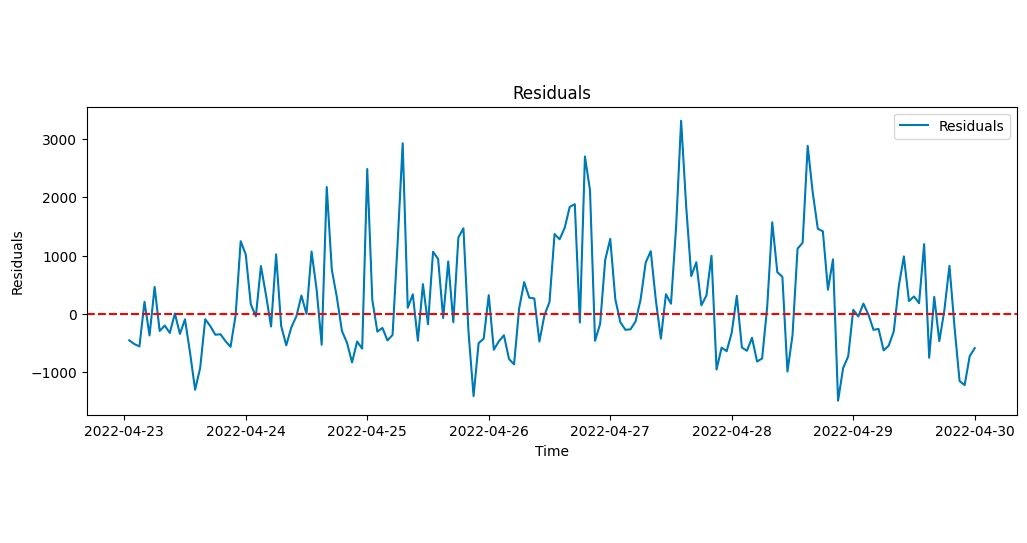


图 3-10 预测结果和实际负载值的残差对比表 3-8 Prophet 模型预测效果指标

|  |  |  |
| --- | --- | --- |
| **指标** | **数值** | **MA 模型基准** |
| MSE | 823243.67 | 1139965.31 |
| RMSE | 907.33 | 1067.69 |
| MAE | 673.04 | 805.27 |
| MAPE | 31.22% | 39.11% |
| SMAPE | 29.23% | 34.47% |

3.2.5 各个模型的预测性能对比

表 3-9 模型预测效果对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **指标** | **MA 模型** | **VAR 模型** | **ARIMA 模型** | **Prophet 模型** |
| MSE | 1139965.31 | 680996.57 | 291700.66 | 823243.67 |
| RMSE | 1067.69 | 825.23 | 540.09 | 907.33 |
| MAE | 805.27 | 590.93 | 405.24 | 673.04 |
| MAPE | 39.11% | 26.98% | 19.30% | 31.22% |
| SMAPE | 34.47% | 25.54% | 17.82% | 29.23% |

从表 3-9 中我们可观察到，四种模型的预测效果在各个指标上有所不同。ARIMA 模型在所有评测指标上的表现都优于其他三种模型。ARIMA 模型的 MSE、RMSE、MAE、

MAPE 和 SMAPE 分别为 291700.66、540.09、405.24、19.30% 和 17.82%，这些指标的数值均低于其他模型，显示出 ARIMA 模型具有更高的预测精度。

##### VAR 模型的 MSE、RMSE、MAE、MAPE 和 SMAPE 数值为 680996.57、825.23、590.93、

26.98% 和 25.54%，排名次之，也显示出良好的预测性能。然后是 Prophet 模型，虽然其在所有指标上的表现都优于 MA 模型，但是与 ARIMA 和 VAR 模型相比则略显逊色。

最后，MA 模型的预测效果最差，所有评测指标的数值都最高，显示出预测误差较大。

综上，从预测精度角度来看，在本任务中，ARIMA 模型是四种模型中表现最好的，然后是 VAR 模型，Prophet 模型和 MA 模型的预测效果相对较差。

###### **3.2.6** 各个模型的训练时间对比

表 3-10 模型训练时间对比

|  |  |
| --- | --- |
| **模型** | **训练时间 (s)** |
| MA 模型 | 0.0 |
| VAR 模型 | 0.4 |
| ARIMA 模型 | 0.1 |
| Prophet 模型 | 4.7 |

从表 3-10 中我们可观察到，四种模型的训练时间存在显著差异。其中，MA 模型的训练时间最短，几乎为零，表示该模型可以即时进行预测。其次是 ARIMA 模型，训练时间为 0.1s，也是非常快。VAR 模型的训练时间稍长，为 0.4s。

然而，Prophet 模型的训练时间最长，达到了 4.7s。这可能是由于 Prophet 模型在构建过程中包含了更多的复杂度和参数，因此需要更长的训练时间。这可能在实时或者需要快速响应的预测任务中会成为一种限制。

因此，在选择模型时，除了考虑预测精度外，我们还需要根据具体应用场景和需求，考虑模型的训练时间，以达到最佳的性能和效率的平衡。在本研究的场景中，Serverless 流量预测需要滚动训练模型，因此如果涉及的服务较多，历史数据较长，应当在应用时增加对训练时间的考虑权重。

###### **3.2.7** 结论

从上述分析可以得出结论，尽管各模型在预测性能和训练时间上都有各自的优势，但在综合考虑预测精度和训练时间的情况下，ARIMA 模型成为本研究的首选。

首先，ARIMA 模型在所有的评价指标上，包括 MSE、RMSE、MAE、MAPE 和 SMAPE，均表现出最优的预测性能，这说明 ARIMA 模型在预测准确性上胜过其他模型。预测精度是我们选用预测模型的重要依据，因为高精度的预测结果可以为伸缩决策提供更可靠的依据，减小决策的风险。

其次，ARIMA 模型的训练时间仅为 0.1s，虽然不及 MA 模型的瞬时训练，但仍远低于 Prophet 模型的 4.7s，也比 VAR 模型的 0.4s 要短。在许多情况下，应当模型能够快速地进行训练和预测，以适应数据的实时变化和快速响应的需求。

因此，综合预测精度和训练时间，ARIMA 模型被我们选择为主要的预测模型。

## 第四章 算法的部署与应用

OpenFaaS 是一个开源的函数即服务（Function as a Service，FaaS）平台，它通过将业务逻辑封装成可重用、独立的函数来提供服务。相比于传统的服务器架构，FaaS 更加轻量级、弹性化，具有更好的可扩展性和更低的成本。

本章将介绍如何在 OpenFaaS 平台上部署本文设计的算法，并通过一个实际的案例来展示该算法的应用效果。

### **4.1** 整体设计

本研究所提出的系统设计以高效性、稳定性和自适应性为目标，围绕算法模块、监控模块和控制模块进行布局。下面，我们将详细阐述每个模块的设计思路和实现方式。

图 4-1 展示了整体设计思路。

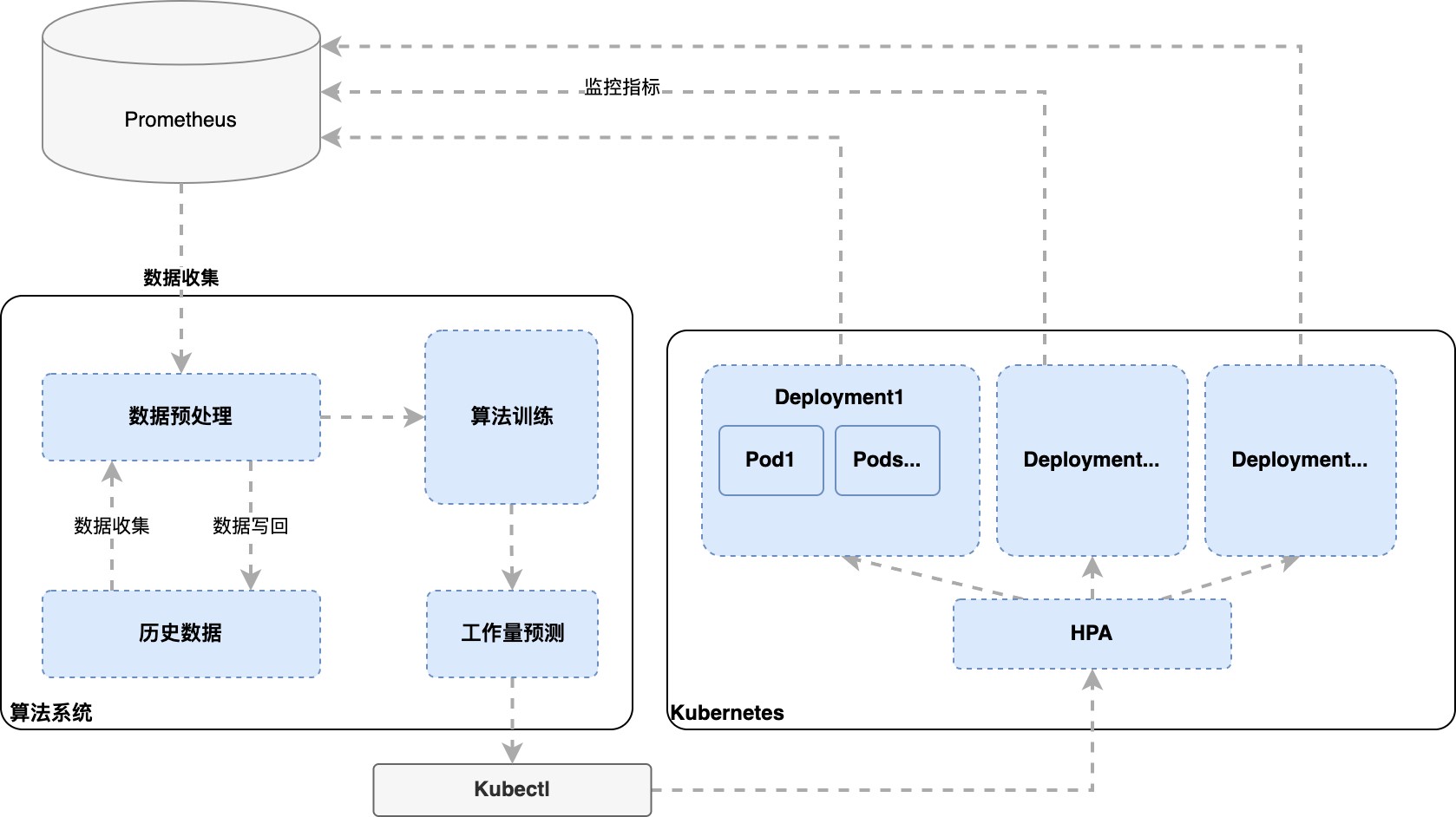


图 4-1 部署设计图

#### 4.1.1 算法

算法模块是整个系统的核心，包括数据处理和训练两个子模块，负责模型的构建、训练和预测。该模块可通过手动调用或定时任务启动，对系统内的大量信息进行实时处理和决策，以实现自动化的负载调度和资源分配。

在系统启动之后，算法模块会通过调用 Prometheus 监控服务的 Web API 收集过去一段时间 T 的调用情况，并将这些数据与历史数据合并，构成训练样本，输入到训练子模块。经过一系列计算，模型在数秒内完成实时训练，并生成下一时间 T+1 的预测值。为了进一步实施负载调度，我们引入预定义的系数 k，将预测值转换为下一时间 T + 1 的副本数量。

#### 4.1.2 监控

监控模块主要负责收集和存储系统的运行数据，提供给算法模块进行处理。这个模块由 Prometheus 提供，其指标来源于 OpenFaaS 容器。通过 Prometheus 的监控，我们可以对系统进行全方位的观察和分析，以便在必要时进行调整和优化。

#### 4.1.3 控制

控制模块是系统的执行部分，负责将算法模块的决策转化为具体的行动。一旦算法模块完成计算并产生预测值，控制模块就会调用 kubectl 的有关命令，将预测值作为目标 Serverless 函数的最小副本数，进行伸缩。这样就可以根据实际需求，动态调整系统的负载和资源分配。

### **4.2** 硬件与软件环境

该实验使用的操作系统为Ubuntu20.04.2LTSx86\_64，主机使用的是OpenStackNova 13.2.1-20230408163250\_be95288。内核版本为 Linux 5.4.0-66-generic。CPU 为 Intel Xeon

Gold 6278C（8 核），主频为 2.600GHz。物理内存大小为 32116MiB。GPU 使用的是

NVIDIA Tesla T4。

### **4.3 Kubernetes** 集群的搭建和 **OpenFaaS** 的部署

为应用本算法，应当搭建一个 Kubernetes 集群，并在该集群中安装 OpenFaaS 以及本文设计的程序。

#### 4.3.1 OpenFaaS 的部署

本节将介绍如何在本地部署 OpenFaaS 平台并测试其功能[12]。

首先，应当安装 Docker 和 Kubernetes。

步骤 1: 首先, 应当在本地环境中安装 Docker 和 Kubernetes 这两个相关的容器组件技术。

步骤 1.1: 要安装 Docker, 应当按照 Docker 官方文档的指引, 在 Linux 主机上执行以下命令, 以便从 Docker 的软件源安装最新版本的 Docker Engine - Community:

|  |
| --- |
| curl -sLS https://get.docker.com | sudo sh |

1

步骤 1.2: 然后, 应当下载 Kubernetes 命令行工具 kubectl 的最新版本。可以按照 Kubernetes 官方文档的说明, 在 Linux 主机上执行以下命令:

|  |
| --- |
| curl -LO "https://dl.k8s.io/release/$(curl␣-L␣-s␣ https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"  sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl kubectl version --client |

1

2

3

步骤 2: 安装完 Docker 和 kubectl 之后, 应当安装 Kubernetes 的本地虚拟集群工具 Kind。可以从项目的 GitHub 发行版页面下载 Kind 的最新版本, 然后执行以下命令进行安装:

|  |
| --- |
| wget "https://github.com/kubernetes-sigs/kind/releases"\  "/download/v0.18.0/kind-linux-amd64" -O kind chmod +x kind sudo mv kind /usr/local/bin/ |

1

2

3

4

步骤 3: 使用 Kind 工具创建一个名为”openfaas” 的 Kubernetes 本地集群:

|  |
| --- |
| kind create cluster --name openfaas |

1

步骤 4: 我们继续执行 Kind 和 kubectl 命令设置当前工作环境变量以操作该 openfaas 集群:

|  |
| --- |
| export KUBECONFIG="$(kind␣get␣kubeconfig-path␣--name="openfaas")" |

1

步骤 5: 最后, 我们使用 arkade 这一自动化工具安装 OpenFaaS 组件到集群之上。我们按照 arkade 的安装指引, 在 Linux 主机上下载并执行其安装脚本:

|  |
| --- |
| curl -sLS https://get.arkade.dev | sudo sh arkade install openfaas |

1

2

以上命令将自动安装所需的软件包并创建名为openfaas的 Kubernetes 集群，最后使用arkade工具在集群中安装 OpenFaaS。安装完成后，使用以下命令来检查 OpenFaaS 是否成功安装：

|  |
| --- |
| kubectl get pods -n openfaas |

1

可以看到，OpenFaaS 的各个组件均已正常运行。

##### **4.3.1.1** 部署函数

在 OpenFaaS 中，函数是以 Docker 容器的形式进行部署和管理的。使用 faas-cli 工具可以方便地进行函数的部署。对于本文的示例，可以使用以下命令将 nodeinfo 函数部署到 OpenFaaS 中：

|  |
| --- |
| faas-cli store deploy nodeinfo |

1

这个命令的作用是从 OpenFaaS 的函数商店中拉取 nodeinfo 函数的镜像，并将其部署到 OpenFaaS 环境中。

##### **4.3.1.2** 调用函数

函数成功部署到 OpenFaaS 环境后，可以使用 HTTP 请求来调用。以下命令向部署在本地的 OpenFaaS 环境中的 nodeinfo 函数发送 HTTP GET 请求：

|  |
| --- |
| curl -v http://127.0.0.1:8080/function/nodeinfo |

1

在接收到请求后，OpenFaaS 将会将请求转发给部署的 nodeinfo 函数，并将函数的输出返回给客户端。

##### **4.3.1.3** 查看部署状态

使用 kubectl 命令可以查看 OpenFaaS 环境中部署的函数的状态。使用以下命令查看 nodeinfo 函数的部署状态[13]：

|  |
| --- |
| kubectl get deployments -n openfaas-fn |

1

命令返回一个包含了所有 OpenFaaS 环境中部署的函数的列表。对于本文的示例，可以看到 nodeinfo 函数已经被成功部署。

此外，也可以使用以下命令查看 nodeinfo 函数所在的 Pod 的状态：

|  |
| --- |
| kubectl get pods -n openfaas-fn |

1

该命令将返回一个包含了所有 OpenFaaS 环境中正在运行的 Pod 的列表。对于本文的示例，可以看到 nodeinfo 函数所在的 Pod 正在运行。

### **4.4 Prometheus** 获取指标

Prometheus 提供了 HTTP API 来执行各种操作，包括获取时间序列数据、元数据等[14]。为了给算法提供数据，应当使用 Prometheus 的 HTTP API 来获取 OpenFaaS 环境中的函数的相关指标。具体思路为，获取最近一个小时的调用次数，作为数据点加入到数据集中，并快速训练出对应的 ARIMA 模型，生成预测数据点，然后根据预测数据点进行伸缩。

获取最近一小时的数据，需使用 range queries API。

|  |
| --- |
| curl "http://localhost:9090/api/v1/query\_range"\  "?query=gateway\_function\_invocation\_total"\  "&start=<start\_timestamp>"\  "&end=<end\_timestamp>"\  "&step=<step\_seconds>" |

1

2

3

4

5

从而得到一个包含所需数据的响应：

|  |  |
| --- | --- |
| { | "status": "success", |

1

2

|  |  |
| --- | --- |
| } | "data": {  "resultType": "matrix", "result": [ {  "metric": {  "\_\_name\_\_": "gateway\_function\_invocation\_total",  "instance": "localhost:9090",  "job": "prometheus"  },  "values": [ ...  [1681347300, "220"]  ]  }  ]  } |

3

4

5

6

7

8

9

10 11

12

13

14

15

16

17

18

19

**4.4.1** 根据预测结果调整副本数量

#### 在 HTTP API 中，可以使用 POST /system/functions 接口来部署或更新函数，包括调

整副本数量。为了调整 nodeinfo 函数的副本数量，控制程序会发送以下 HTTP 请求

|  |
| --- |
| POST /system/functions HTTP/1.1  Host: <OpenFaaS Gateway URL>  Content-Type: application/json  Authorization: Bearer <Your OpenFaaS Token>  {  "service": "nodeinfo",  "image": "functions/nodeinfo",  "labels": {  "com.openfaas.scale.min": "2",  "com.openfaas.scale.max": "5" }  } |

1

2

3

4

5

6

7

8

9

10 11

12

13

在 labels 字段中，com.openfaas.scale.min 和 com.openfaas.scale.max 分别表示最小和最大的副本数量。除此之外，也可以通过 kubectl 直接设置副本的数量：

|  |
| --- |
| kubectl scale deployment nodeinfo -n openfaas-fn --replicas=5 |

1

然后，OpenFaaS 控制器将会根据副本数量的变化，自动调整 nodeinfo 函数的副本数量。可以通过如下命令检查 nodeinfo 函数的副本数量：

|  |
| --- |
| kubectl get deployment nodeinfo -n openfaas-fn -o=jsonpath='{.spec.replicas}' |

1

##### **4.4.2** 自动伸缩：**HPA** 控制器

在容器编排中，HPA（Horizontal Pod Autoscaler）用于根据负载自动调整 Pod 的数量。

Deployment 是用于定义和管理 Pod 副本的资源对象。Pod 则代表着运行在 Kubernetes 集群中的一个实例化容器[15]。

图 4-2 中的箭头表示了对象之间的关系。箭头从 HPA 指向 Deployment，表示 HPA 通过扩展或收缩 Deployment 来调整 Pod 的数量。Deployment 和 Pod 之间的箭头表示

Deployment 控制着多个 Pod 的创建和管理。

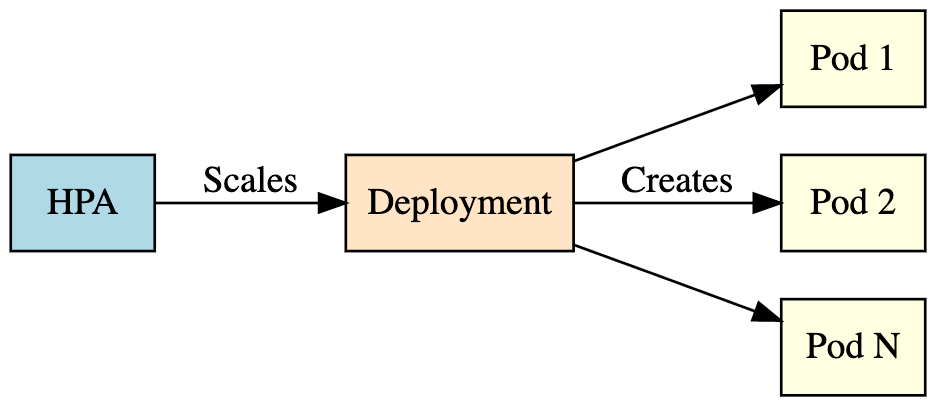


图 4-2 HPA 控制器原理

#### 4.5 流量模拟与伸缩实验

##### **4.5.1** 请求变化曲线的构造

为了验证自动伸缩，首先需要生成流量的变化曲线。由于实际使用时伸缩的周期是数小时，为了方便验证，我们将时间尺度缩小到数分钟。然后，我们选用一个简单的函数来模拟请求流量的变化曲线。

具体而言，这个函数充当一个数据生成器，它可以根据给定的种子和长度，生成具有特定属性的时间序列数据。我们希望这些数据具有一定的随机性，以模拟实际场景的不确定性；同时，我们还希望它具有一定的周期性（包括短周期和长周期），以反映某些周期性因素的影响；此外，我们希望这些数据具有整体的上升趋势，从而反映某些长期趋势（例如业务增长）的影响。最后，我们希望这些数据是整数，并且其值在 0 1000 之间，从而确保它们与实际的函数调用次数具有相同的数量级。

数据生成器函数的代码如下：

|  |
| --- |
| import numpy as np import matplotlib.pyplot as plt  def generate\_data(seed, t\_length, small\_period=24, large\_period=720):  np.random.seed(seed)  # 初始化序列  y = np.zeros(t\_length)  # 生成随机步长。此处，我们使步长的平均值稍微偏正，以保证整体的升趋势。  steps = np.random.randint(-40, 45, size=t\_length - 1)  # 计算随机漫步序列  for t in range(1, t\_length):  y[t] = y[t-1] + steps[t-1]  # 添加周期性趋势  t = np.arange(t\_length)  y += np.sin(2 \* np.pi \* t / small\_period) \* 50 # 小周期趋势 y += np.sin(2 \* np.pi \* t / large\_period) \* 100 #  大周期趋势，可以调整振幅以增加或减少周期性趋势的强度  # 为了保证y的范围在0~1000之间，我们将y线性映射到这个范围  y = (y - np.min(y)) / (np.max(y) - np.min(y)) \* 1000  # 因为y应该是整数，所以我们进行四舍五入  y = np.round(y).astype(int)  return y |

1

2

3

4

5

6

7

8

9 10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

它的工作原理如下：

首先，我们使用给定的种子初始化随机数生成器，以确保本研究的数据生成过程是可复现的。然后，我们生成一个随机步长序列，这个步长序列可以是正的或负的，但我们使其平均值稍微偏正，以保证整体的上升趋势。然后，我们使用这个步长序列生成一个随机漫步序列，这个随机漫步序列体现了数据的随机性。

接下来，我们添加了两个周期性趋势。一个是小周期趋势，周期为 24，模拟了一天内的变化；另一个是大周期趋势，周期为 720，模拟了一个月内的变化。我们使用 sine 函数来生成这两个周期性趋势，因为 sine 函数是周期性的，而且它的值在-1 和 1 之间，这使得可以通过调整振幅来控制周期性趋势的强度。

最后，为了保证生成的数据的值在 0 1000 之间且是整数，我们将随机漫步序列和周期性趋势的和线性映射到这个范围，然后进行四舍五入。线性映射的过程是通过计算最小值和最大值，然后使用这两个值将数据缩放和平移到目标范围。

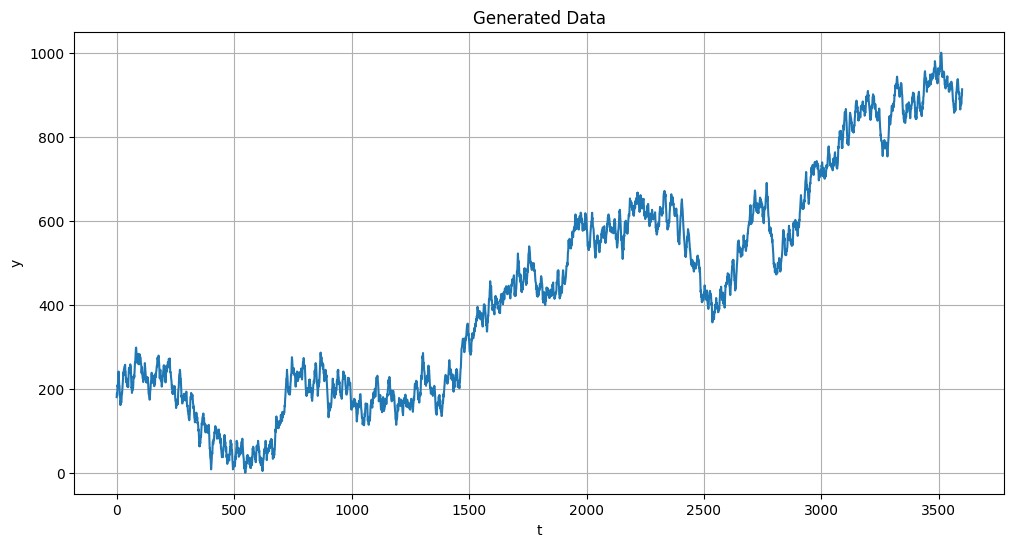


图 4-3 生成的数据示例

生成的数据的示例如图 4-3 所示。

##### **4.5.2** 基于模拟流量的自动伸缩仿真

我们使用 0 到 999 时刻的数据作为初始历史数据，输入到 ARIMA 模型训练，并预测 1000 时刻的访问量。从 1000 时刻开始，每隔 30 秒，采用下一个时刻的数据点作为模拟的请求数量。并滑动历史数据窗口，将新的数据点加入到模型训练的历史数据集中，以模拟实际的工作负载预测过程。

基于预测访问量，使用请求/容器数量的比例（记作 k）来计算需要扩容至的容器数量。此处我们采用 k=100，也即每个容器每秒处理 100 个请求的假设。此假设仅用于演示，其取值不影响预测性能或伸缩效率。在实际应用时，应当根据服务器配置情况测量得到此参数。

图 4-4 展示访问量与副本数量的关系，即假设伸缩不消耗时间情况下副本的数量关系。

##### **4.5.3** 仿真结果与结果分析

使用上述结果进行仿真，对于预测区间 [1001,2000] ，ARIMA 模型得到的结果如图 4-5 所示。作为基准的 MA（WIN\_SIZE=3）模型得到的结果如图 4-6 所示。这两张图

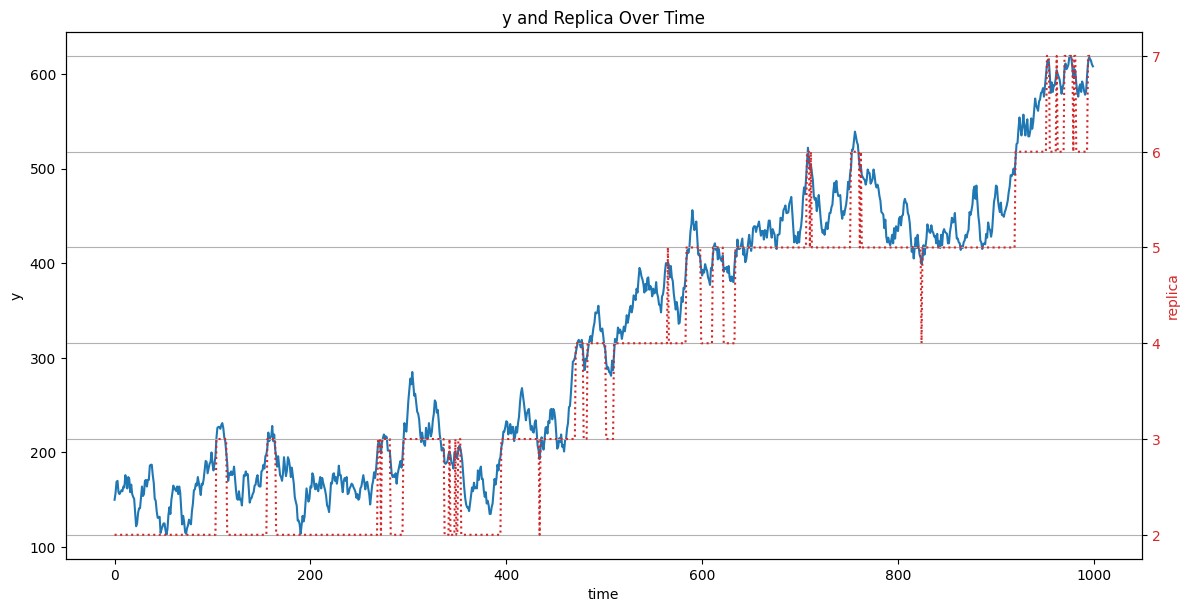


图 4-4 访问量与副本数量

展示实际值（蓝色曲线）与预测值（绿色曲线）之间的关系，同时使用右侧坐标轴展示实际所需副本数和预测所需副本数之间的关系。

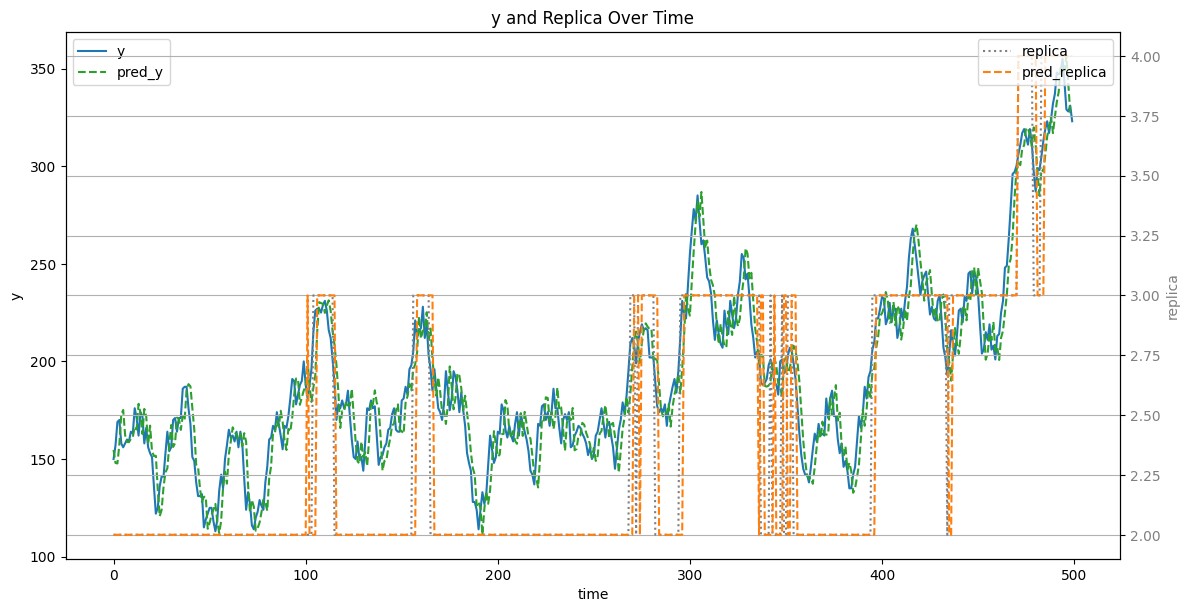


图 4-5 ARIMA 模型仿真结果

表 4-1 展示模型的性能指标。从 MAPE、SMAPE 的结果来看，模型的预测性能较好。且 ARIMA 模型的性能在所有指标上超过 MA 模型。

而副本数量预测任务，由于相当于对访问量预测进行向上取整，预测的性能表现更为优异，如表 4-2 所示。副本数量的预测几乎完全准确。且 ARIMA 模型的性能在所有指标上均优于 MA 模型。

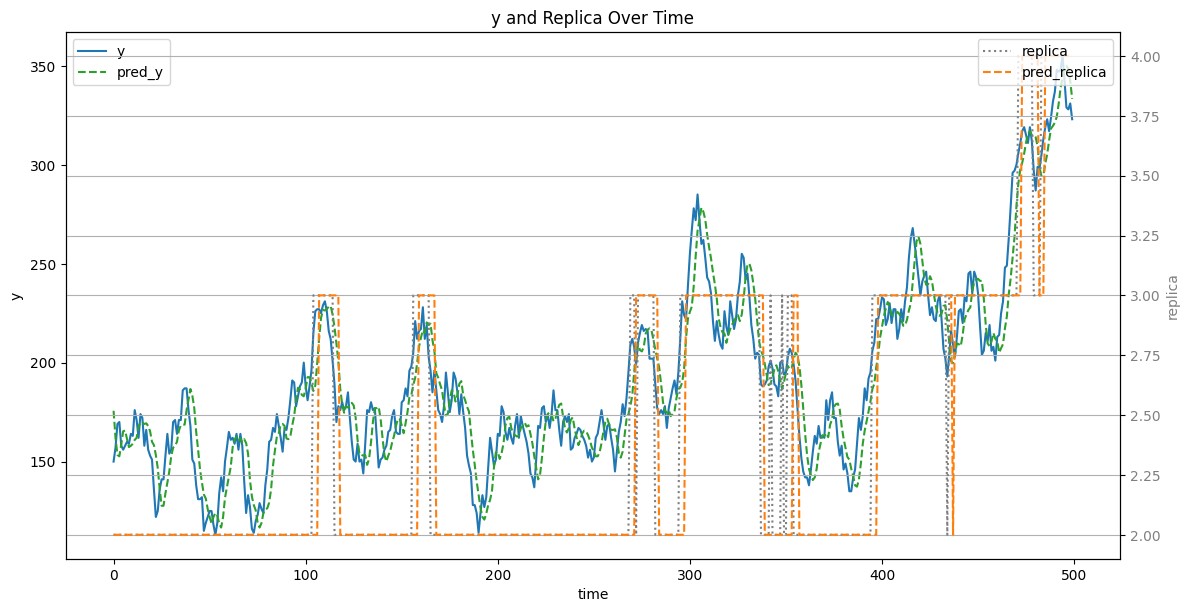


图 4-6 MA 模型仿真结果

表 4-1 基于仿真数据预测的模型性能

|  |  |  |
| --- | --- | --- |
| **指标** | **数值** | **MA 模型基准** |
| MSE | 169.09 | 243.23 |
| RMSE | 13.00 | 15.60 |
| MAE | 10.68 | 12.72 |
| MAPE | 5.78 % | 6.89% |
| SMAPE | 5.78 % | S 6.87% |

表 4-2 基于仿真数据预测的模型性能（副本数量）

|  |  |  |
| --- | --- | --- |
| **指标** | **数值** | **MA 模型基准** |
| MSE | 0.074 | 0.086 |
| RMSE | 0.27 | 0.29 |
| MAE | 0.074 | 0.086 |
| MAPE | 2.93 % | 3.30% |
| SMAPE | 2.86 % | 3.28% |

## 第五章 结论与展望

### **5.1** 结论

#### 5.1.1 本文的主要成果

本研究中提出并实施了一种基于负载预测的 Serverless 计算的自动伸缩算法。主要发现如下：

* 我们基于 MA、VAR、Prophet 和 ARIMA 模型实现了有效的流量预测机制，从而能够在工作负载发生变化之前做出响应，提前进行自动伸缩。通过对实际的 Serverless 工作负载进行实验，我们发现这些模型能够有效预测其变化趋势。
* 我们设计并实施了一套基于 Prometheus 的自动预警机制。通过收集 Serverless 应用的监控指标，并将这些数据输入到预测模型中，可以提前做出伸缩决策，从而避免因工作负载变化导致的性能下降或资源浪费。
* 我们还实现了函数副本数量的自动伸缩。通过调整副本数量，Serverless 应用可以

根据实际的工作负载需求动态调整资源使用，从而提高资源利用率和运行稳定性。

值得一提的是，我们在每个预测时刻之前，会采用滑动窗口的方式重新训练模型，并且自动选择最优的 p,d,q 参数。这主要有两点好处：一是可以避免历史数据膨胀导致预测的时间复杂度常数过高；二是可以避免模型参数无法同步于历史数据的变化导致预测结果的不准确性。

#### 5.1.2 对实际问题的解决方案

本研究所提出的自动伸缩算法和方案，为如何实现 Serverless 计算的自动伸缩提供了有效的解决方案。通过自动预警、负载预测、自动伸缩的结合，可以在工作负载变化之前做出响应，动态调整函数副本数量，提高资源利用率和运行稳定性。

#### 5.2 局限性与未来工作

##### **5.2.1** 本文的不足之处

虽然本研究取得了一些重要的发现，但还存在以下不足之处：

* 本研究的负载预测模型基于历史数据进行预测，但对于那些历史数据不足，预测准确性可能会降低。
* 我们发现多数的 Serverless 应用，即便在日尺度下有非常高的访问次数，但在分钟或秒的尺度之下，统计周期内可能根本没有访问，所以对于实际场景，精确到秒的预测的难度会非常大。这也是我们选择以小时为单位的原因。
* 允许零副本的情况下，本研究的自动伸缩算法由于采用向上取整，最少的副本数量为 1，这可能会导致资源浪费。
* 我们实验时发现特定时间点负载的变化幅度较大，导致频繁的伸缩行为，这可能会影响应用的性能。

##### **5.2.2** 可进一步探讨和优化的方向

针对上述不足之处，未来的研究可以从以下方向进行：

* **使用预训练模型进行预测。**可以考虑使用 Transformer 等模型，在同类 Serverless 应用上进行预训练，再使用具体的应用进行微调，以提高预测的准确性[16]。
* **增加更多的预测特征。**目前主要针对时间和访问次数进行预测，未来可以考虑引入更多内部和外部的特征，例如，CPU 使用率，内存使用率，网络延迟等，以提高预测的准确性。不过这主要适用于多特征的预测模型。
* **考虑支持零副本的缩容。**对于重要度不高，能接受数秒的启动延迟的应用，可以考虑支持零副本的缩容，从而避免资源浪费。
* **考虑增加对临界值的特殊处理。**预测值量变临界情况下，可以增加一定的迟滞性，从而避免过于频繁的伸缩。

这些方向将为我们提供更深入的理解和优化 Serverless 计算的自动伸缩。

## 参考文献

1. Jumpertz Oliver. How AWS Lambda Works Under The Hood [EB/OL]. 2020. [https://oliverjumpertz.c om/how-aws-lambda-works-under-the-hood/](https://oliverjumpertz.com/how-aws-lambda-works-under-the-hood/).
2. OpenFaaS Org. OpenFaaS Design & Architecture: Gateway [EB/OL]. 2023 [2023-4-10]. [https://docs .openfaas.com/architecture/gateway/](https://docs.openfaas.com/architecture/gateway/).
3. Rizkya Indah, Syahputri Khalida, Sari Rachida et al. Autoregressive Integrated Moving Average (ARIMA) Model of Forecast Demand in Distribution Centre [J]. IOP Conference Series Materials Science and Engineering. 598 (1). 2019, September: 012–071.
4. Hyndman Rob J, Athanasopoulos George. Forecasting: Principles and Practice [M]. OTexts, 2018.
5. Taylor Sean J, Letham Benjamin. Forecasting at Scale [J]. The American Statistician. 72 (1). 2018: 37–45.
6. Robert Christian, Casella George. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data [J/OL]. Statistical Science. 26 (1). 2011: 102–115. [http://www.jstor.or g/stable/23059158](http://www.jstor.org/stable/23059158).
7. TranMinh-Ngoc, NguyenTrong-Nghia, DaoViet-Hung.ApracticaltutorialonVariationalBayes.2021.
8. Akaike H. This Week’s Citation Classic [J]. Current Contents Engineering, Technology, and Applied Sciences. 12 (51). 1981: 42. Hirotogu Akaike comments on how he arrived at AIC.
9. Schwarz Gideon E. Estimating the dimension of a model [J]. Annals of Statistics. 6 (2). 1978: 461–464. MR 0468014.
10. Niedźwiecki Maciej, Ciołek Marcin. Akaike’s final prediction error criterion revisited [C]. In 2017 40th International Conference on Telecommunications and Signal Processing (TSP). 2017 : 237–242.
11. Pollock D S G. Handbook of Time Series Analysis, Signal Processing, and Dynamics [M]. har/cdr ed. Academic Press, 1999.
12. Ellis Alex. Get started with OpenFaaS and KinD [EB/OL]. 2019. [https://blog.alexellis.io/get-startedwith-openfaas-and-kind/](https://blog.alexellis.io/get-started-with-openfaas-and-kind/).
13. Authors Kubernetes. Kubernetes Deployment [EB/OL]. 2023. [https://kubernetes.io/docs/concepts/wo rkloads/controllers/deployment/](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/).
14. Authors Prometheus. Prometheus Querying API [EB/OL]. 2023. [https://prometheus.io/docs/prometheu s/latest/querying/api/](https://prometheus.io/docs/prometheus/latest/querying/api/).
15. Authors Kubernetes. Kubernetes HPA [EB/OL]. 2023. [https://kubernetes.io/zh-cn/docs/tasks/run-appli cation/horizontal-pod-autoscale/](https://kubernetes.io/zh-cn/docs/tasks/run-application/horizontal-pod-autoscale/).
16. Wu Neo, Green Bradley, Ben Xue et al. Deep Transformer Models for Time Series Forecasting: The Influenza Prevalence Case [J]. arXiv preprint arXiv:2001.08317. 2020: 1.