同濟大學

TONGJI UNIVERSITY

毕业实训

课题名称 基于微服务相关性的负载预测工具设计与开发

副 标 题 _____

学　　院 电子与信息工程学院

专　　业 _____

学生姓名 _____

学　　号 _____

指导教师 _____

日　　期 2021 年 1 月 9 日

摘 要

在微服务容器化的应用部署背景下，为了实时服务于用户的个性化请求，云平台上运行着大量的微服务。由于微服务应用的分布式架构（DAG 图结构），微服务的资源使用量（资源负载）具有了显著的相关性关系，主要体现在两个方面：一是基于微服务之间调用或事件触发关系的不同微服务之间的资源负载相关性；二是基于负载均衡技术产生的相同微服务不同实例之间的资源负载相关性。因此，充分挖掘微服务负载之间的相关性关系将有助于资源负载变化的分析及预测。

本课题旨在开发一个基于微服务相关性的在线资源负载预测工具。提出基于深度学习模型考虑微服务相关性的资源负载预测方案；基于已有的面向 K8S 的微服务负载预测平台，开发基于微服务相关性的资源负载预测工具并集成到系统当中。

本篇报告总结了微服务相关性的研究意义和国内外研究现状，并详细描述了基于微服务相关性的资源负载预测工具系统的设计思路，总框图，模型的设计以及最终运行结果。

关键词：云计算，微服务，负载预测

ABSTRACT

In the context of microservices containerized application deployment, a large number of microservices are running on the cloud platform in order to serve the personalized requests of users in real time. Due to the distributed architecture (DAG graph structure) of microservice applications, the resource usage (resource load) of microservices has a significant correlation relationship, which is mainly reflected in two aspects: first, the resource load correlation between different microservices based on the invocation or event-triggered relationship between microservices; second, the resource load correlation between different instances of the same microservice based on the load balancing technology. Therefore, fully mining the correlation relationship between microservice loads will help the analysis and prediction of resource load changes.

This topic aims to develop an online resource load prediction tool based on microservice correlations. A deep learning model-based resource load prediction scheme considering microservice correlation is proposed; based on the existing K8S-oriented microservice load prediction platform, a microservice correlation-based resource load prediction tool is developed and integrated into the system.

This report summarizes the research significance of microservice relevance and the current status of domestic and international research, and describes in detail the design idea, general block diagram, model design and final operation results of the resource load prediction tool system based on microservice relevance.

Key words：Cloud Computing, Microservice, Load Prediction

装 订 线

目 录

装 订 线

1 引言

1.1 课题背景

云计算是一项很有前途的技术，旨在带来各种可视化的资源、软件和平台并将它们作为服务，以基于使用付费的模式提供给客户。为了给终端用户提供高性能的云服务，在云数据中心中进行资源管理是非常重要的。除了满足客户的基本 QoS（Quality of Service，一种由用户定义的服务需要满足的质量协议）要求以外，合理且有效的资源管理方法还可以减少能源消耗成本和二氧化碳排放量。

一般来说，资源管理方案可以分为反应式和主动式。

在第一种情况下，当工作负载增加/减少到预先定义的阈值时，云数据中心的控制单元将进行资源管理。但是，由于虚拟机的启动时间，容器迁移镜像和保存快照的时间等，被动的资源管理方法不能处理突然爆发的工作负载，并可能导致违反服务水平协议（SLA）。

而第二种情况下，主动式方法通过预测未来的工作量来解决这个问题。主动式方法通过识别可能的资源使用模式来预测数据中心未来的工作量，并提供所需的资源，从而解决这个问题。因此，通过有效的负载预测，可以阻止在未来出现性能下降的异常，并减少闲置资源以进一步提高利润。然而，进行主动的资源管理并不是一个简单的过程，云托管服务的可变工作量可能会导致以下问题：

- 供给不足（Under-provisioning）

  应用程序没有得到足够的资源来处理他们的所有请求，并可能导致违反服务水平协议（SLAV）。

- 超额配置（Over-provisioning）

  虚拟资源的分配超过了应用程序的需要。这给客户带来了更高的成本。在某些情况下，也需要超额配置来处理工作负载的波动。

- 振荡（Oscillation）

  由于自动扩容缩容算法带来的超额配置和配置不足问题的交替出现。

由上述的描述可见，想要实现有效的主动资源管理计划需要一个准确的工作负载预测算法。然而，云计算的工作负载预测是一个具有挑战性的问题。与 HPC 系统和网格计算不同，云计算的工作负载具有更高的变动性，并且是在一个特定的环境下进行的，其平均噪音几乎是网格计算的 20 倍[1]。此外，由于云资源是由多个用户或任务共享的，它们可能会遭受到其他用户操作或服务引起的波动，而且新的工作负载模式也会不断出现。其模式变化，使预测模型的再训练更加频繁，并增加了开销。

考虑到准确的负载预测在云计算 DC 的有效资源管理中的重要性，为了解决这些问题，人们已经研究和应用了各种数学模型和基于机器学习的预测算法。部分模型和算法对于负载预测问题具有启发性的意义，但是它们仍然不能做到有效利用微服务的相关性同时能广泛适

用于不同的微服务架构。

1.2 研究现状

本章节列举了目前关于微服务负载预测研究的经典算法和模型，并按照 A survey and classification of the workload forecasting methods in cloud computing[1]中提到的分类框架进行分类。

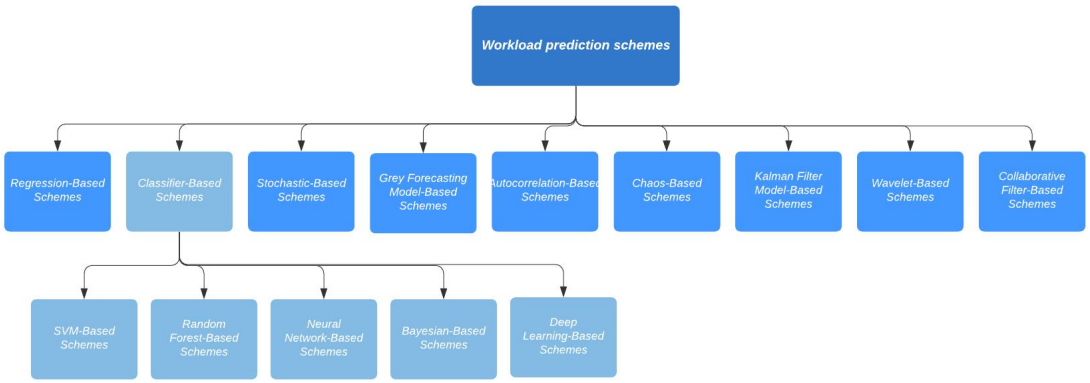

图 1 负载预测研究算法分类图[1]

如上图所示，目前的负载预测算法主要分为 11 类，分别是基于回归的负载预测算法（Regression-Based Schemes），基于分类器的负载预测算法(Classifier-Based Schemes)，基于随机过程的负载预测算法(Stochastic-Based Schemes)，基于模型的灰度预测算法(Grey Forecasting Model-Based Schemes)，基于自我相关函数的负载预测算法(Autocorrelation-Based Schemes)，基于噪音的负载预测算法(Chaos-Based Schemes)，基于模型的卡尔曼滤波负载预测算法(Kalman Filter Model-Based Schemes)，基于小波的负载预测算法(Wavelet-Based Schemes)，基于协同过滤的负载预测算法(Collaborative Filter-Based Schemes)，基于集成的负载预测算法(Ensemble-Based Schemes)，混合方法的预测算法(Hybrid Schemes)等。

就基于回归的负载预测算法这一类而言，Alexandru-Florian Antonescu 提出了两种面向 SLA(Service Level Agreements，一系列关于系统性能和服务质量的严格要求)的虚拟机资源缩放算法[2]。该算法是在基于常量负载的数据集上，结合 SLA 约束学习得来的，使用了面向 SLA 的自回归预测模型。作者使用了诸如 RMSD，执行时间，虚拟机数量等全面的评估指标，并验证了该算法可以保证分布式应用的性能，优于响应式的面向 SLA 的资源缩放算法。Yang 提出了一个线性回归模型来估计负载并将其应用于自动缩放资源的场景中来实现实时的缩放和主动式的资源缩放[3]。在主动式的资源缩放场景中，他们引入了贪心的方法来获得较低的成本和 SLAV(Service Level Agreements Violation)次数。除此之外，还有部分论文采用了 ARIMA(Autoregressive integrated moving average，差分整合移动平均自回归模型)

回归模型和基于支持向量回归的模型等。

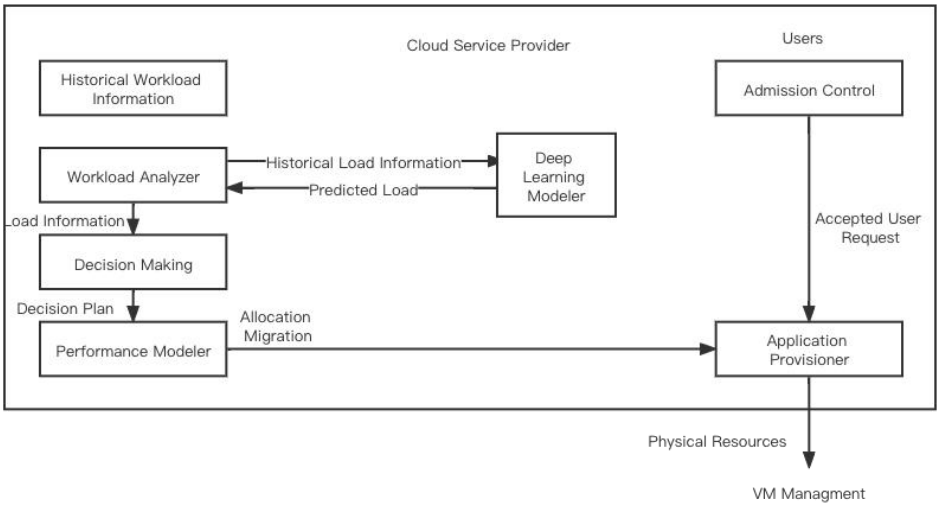

图 2 基于深度学习的面向利用率的负载预测分类器模型结构[9]

就基于分类器的负载预测算法这一类而言，常见的负载预测分类模型有基于支持向量机的分类模型，基于随机森林的分类模型，基于神经网络的分类模型，基于贝叶斯的分类模型和基于深度学习的分类模型等。通常来说，基于分类器的负载预测算法包含了数据的预处理和负载预测两个阶段，数据预处理阶段会执行负载标准化和自相关系数分析等操作[4][5][6]。这些基于分类器的负载预测算法能够有效预测部分场景下的负载，但遗憾的是，部分模型更适用于周期性波动或周期性增加的负载，部分模型则更适用于预测周期性较弱的负载[6]，甚至部分基于随机森林的分类模型并不考虑负载的波动[7]，缺少一个适用于所有场景的通用模型。部分基于神经网络的模型虽然能够针对不同的应用学习不同的特征，并学习不同的资源向量之间的相关性[8]，但是由于神经网络的特点，模型的可解释性较低，并不能证明模型有效地学习到了微服务之间的相关性。基于深度学习的分类模型更适合长期的负载预测场景，模型的性能也可以通过增加训练集的大小和增加模型的深度来进一步地提高。常见的基于深度学习的负载预测模型多利用 CPU 的利用率作为输入指标[9]，以 LSTM 模型[10]或其他时序模型的变体作为基本模型架构。

基于随机过程的负载预测算法多采用马尔可夫模型及其变体。马尔科夫链是一种数学工具，用于模拟一个系统根据某些概率规则，从一个状态过渡到另一个状态的过程。马尔科夫链可以分为离散时间和连续时间马尔科夫链。在[11]中，Pacheco-Sanchez S 利用马尔可夫决策过程及其相关 M/M/1 排序模型来预测已部署的服务的性能。马尔可夫决策模型可以被用在排序模型中预测长尾分布的 HTTP 网络场景下的系统性能。

有关基于模型的灰度预测算法研究较少，在[12]中，Jheng JJ 在负载数据跟时刻有关的场景下通过学习不同天同一时刻的负载数据建立模型来判断当前天的负载趋势是递增还是

递减。他们证明了该模型在负载预测时使用的数据较少且能以更节省能源的方式分配虚拟机的资源。但该模型的预测结果是粗粒度的，且只适用于负载数据跟时刻有关的场景。

在[13]中，Kluge F 提出了使用基于集群自相关函数的负载预测方法，他们的模型解决了负载周期性波动场景下近实时预测的问题，但他们并没有解决在自相关函数执行过程中由于不明确的周期引入的数值不稳定问题。

部分基于噪声的负载预测模型具有低成本和满足 Qos(Quality of service)约束诸如最大响应时间等的优势[14]，在[15]中，该类模型还被证明在负载预测问题上优于基于傅立叶变换的时间序列模型。但是该类模型更适用于边缘计算或地理分布敏感的虚拟机分配问题。

基于卡尔曼滤波模型的负载预测算法的结果为弹性云自动缩放机制提供了一个触发器[16]，该模型提高了负载预测的准确率，降低了自动缩放算法带来的延迟，但它仍需要进一步研究以适用更广泛的负载预测场景。

在基于小波的负载预测算法中，研究者们往往通过调整宿主机上负载的上下限阈值，创造负载时间序列来预测负载子序列的趋势[17]。该预测算法的结果往往用作虚拟机的部署、重分配策略，冗余虚拟机资源的释放策略的参考输入。

在[18]和[19]中都使用了基于协同过滤的算法来预测负载模型，但是这种算法只适用于轻量级的负载预测场景[18]或考虑的指标较少[19]的情况，对于微服务数量较多或需要考虑较多指标的负载预测场景而言并不适合。

上述的研究方法或多或少地在某些场景下存在缺陷，为了弥补该不足，部分研究者采用了集成学习的思想，采用了多种方法集成的负载预测框架。例如，Cao 在[20]中就使用了多个模型来搭建成一个集成模型以提高对于 CPU 负载和性能预测的准确率。他们应用了两层的集成模型，包括预测层和集成层，预测优化层使用新的预测实例并去掉那些表现较差的，集成层汇集多个预测层的结果并向预测优化层提供反馈。

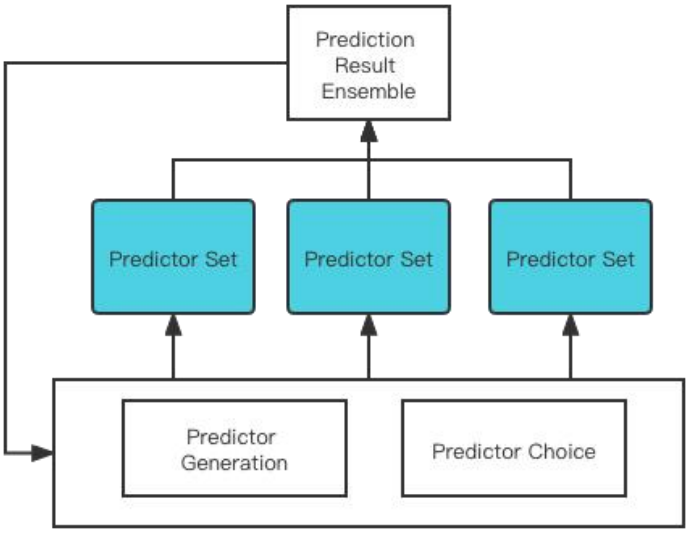

图 3 集成模型架构[20]

2 研究内容与方法

本课题的研究内容主要为基于微服务相关性的负载预测工具的设计开发。用于负载预测的负载及其请求数据一般有两种来源，分别是合成的和真实的数据。合成的负载数据是通过负载生成器来生成的，而真实的数据可以从诸如 Google Cluster trace[21]，NASA-HTTP dataset[22]等数据集中获取。

为了获得不同请求场景下模型的性能和预测结果，同时为了后续毕业设计的主动式部署策略的落地，本课题通过请求负载生成器模拟用户请求并监控 kubernetes[23]集群上应用程序的真实资源消耗情况，预测工具利用历史负载数据构建预测模型，并实时可视化展示负载预测结果。

2.1 负载预测工具的设计与开发

2.1.1 系统设计

2011 年，亚马逊公司提出了云基础架构的自动式管理方案：MAPE 循环[24]。MAPE 是监测、分析、计划和执行(Monitor, Analyze, Plan, Execution)的缩写。MAPE 循环指的是通过重复这四个步骤的循环来实施信息系统操作。信息系统被监控，以检查是否有任何问题发生。如果发现问题，就进行分析以找到原因。制定一个计划来解决问题，并在适当的时候执行解决方案。一个信息系统将反复经历这一过程的 MAPE 循环，旨在尽早发现系统问题并有效地解决它们。本文中的负载预测工具就基于 MAPE 循环理论进行设计，囊括监控模块，分析预测模块，计划决策模块及执行模块。此外，为了便于直观地观察系统资源利用率的变化及预测的有效性，系统另设可视化模块用于用户观测及交互。同时，为了系统后续的可扩展性，系统所有使用到的模型皆存放于模型池中，不同的模块从模型池中获取需要的模型。如后续需要集成其他实验模型，在模型池中添加对应模型并提供相应配置即可。

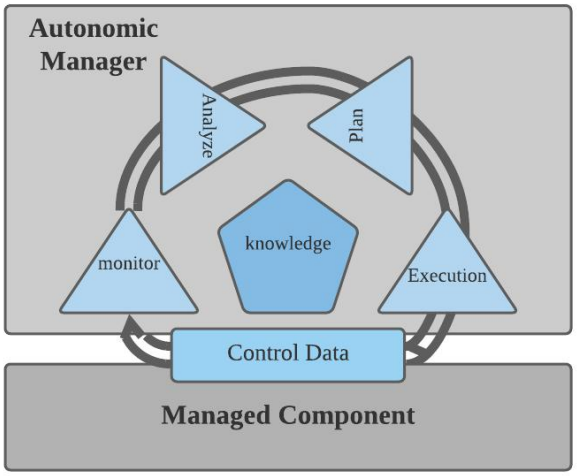

图 4 MAPE 循环

在基本的系统设计中，监控模块主要通过与 kubernetes worker node 上的 cAdvisor 组件进行交互，收集目标资源的实时监控数据汇总成为 Prometheus[25]的评估指标，并以时序数据

的形式存储到数据库。

分析预测模块主要通过读写时序数据库中存储的负载数据并调用模型池中相应的模型来对负载进行预测，并将结果写入到时序数据库中便于可视化模块的读取。

计划决策模块主要通过读写时序数据库中存储的负载数据并调用模型池中相应的模型来生成主动式部署的决策，并将结果传入执行器模块进行决策的落地，该部分的内容主要需在毕业设计阶段完成。

执行模块负责读取计划决策模块生成的决策，并通过调用自定义的 kubernetes 部署执行器来执行部署决策的落地。该部分的内容主要需在毕业设计阶段完成。

在图 5 系统泳道图中将系统的所有运行逻辑分成了监控、预测分析、计划决策、执行、可视化 5 个阶段，用户、前端、后端 3 个职能。

在监控阶段，由用户请求查看系统状态，前端展示系统状态并向后端发出相应的请求监控消息，后端开启相应 Pod 的监控进程并负责维护该监控进程的状态，将监控获得的数据汇总写入时序数据库中。

在可视化阶段即已经开启了监控进程的阶段，前端会与后端建立 websocket 双向链接，由后端向前端推送实时状态并更新前端的展示界面。同时在可视化阶段，用户可以选择切换监控或预测模型。

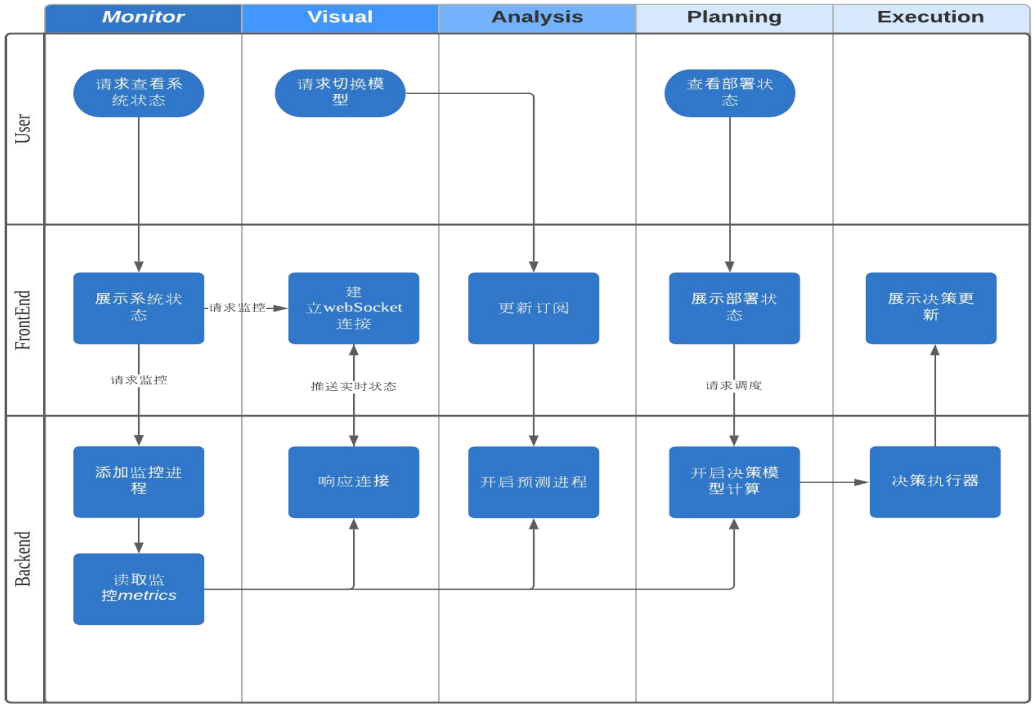

图 5 系统泳道图

在分析预测阶段，前端响应用户选择的分析预测模型，更新对后端 websocket 推送消息的订阅，后端则负责终止过时的监控或预测进程，在模型池中抓取新的模型开启新的监控或预测进程，并推送实时状态到前端。

在计划决策阶段，用户请求查看部署状态，由前端展示部署状态，并向后端发送请求调

装 订 线

度的消息，后端从模型池中抓取相应的决策模型根据当前的状态计算合适的决策并调用决策执行器。

在决策执行阶段，后端启动决策执行器，并推送决策完成后的状态到前端进行展示。

2.1.2 系统部署

图 6 为系统的物理部署图，该图展示了系统的物理架构。系统主要由三个核心组件构成，分别是前端，后端和 kubernetes 集群，用户通过客户终端与前端进行交互。

其中，前端和后端部署在 Master 机器上的不同端口，保证了前后端交互的时延不受网络延迟的影响。Kubernetes 集群运行在特定服务器上，后端通过 kubernetes client API 与 kubernetes 集群进行交互。

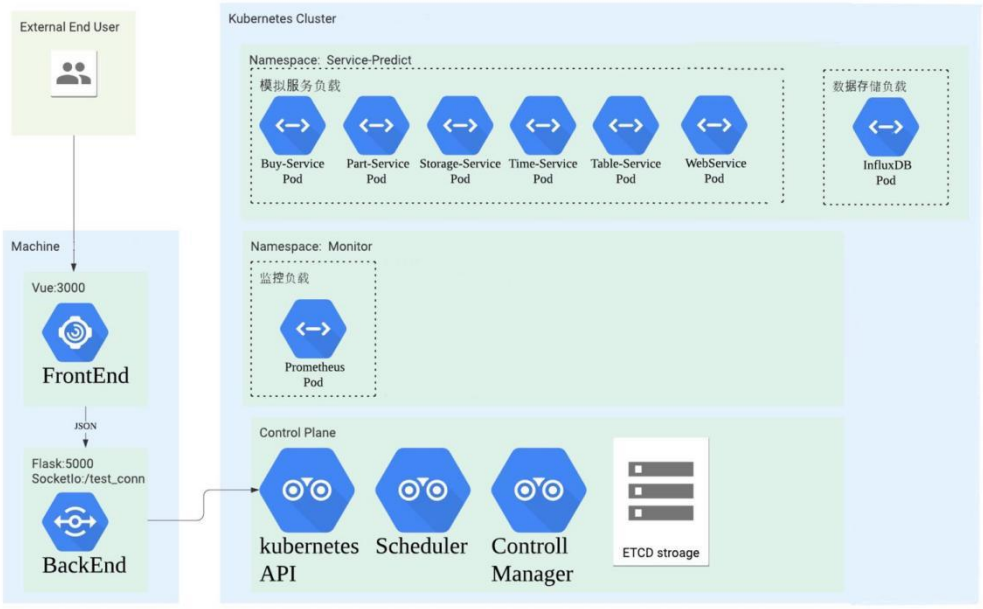

图 6 系统物理部署图

kubernetes 集群上主要运行着三种功能的微服务，分别是模拟负载的微服务，负责时序数据存储的微服务以及负责监控集群实时运行状态的微服务。

另外，在 kubernetes 的 master node 上运行着 kubernetes 的控制面板，该控制面板包括了与后端进行交互的 kubernetes API，负责 Pod 的部署和重部署的调度器 Scheduler，负责控制 worker node 上的 kubelets 的控制器 control manager，以及负责存储 kubernetes 集群上资源状态的分布式数据库 ETCD。

2.1.3 监控模块

监控模块是获取集群相关信息和负载相关数据的核心模块。在获取集群相关信息这一方面，主要采用 Kubernetes 官方的 Python client 包[26]进行开发。后端在进行了目标集群域名，ip 地址，端口的配置之后，通过调用 kubernetes-client 中的方法对运行中的目标 kubernetes 集群进行读写。监控模块可以获得的基本集群信息包括集群配置中的 Nodes，Services，正在运行的 Pods 和相应配置 Deployments。

对于基本的集群信息，用户可在前端选择对应的页面，前端访问后端相应的 API 端口，

后端以 json 格式的形式返回相应信息。

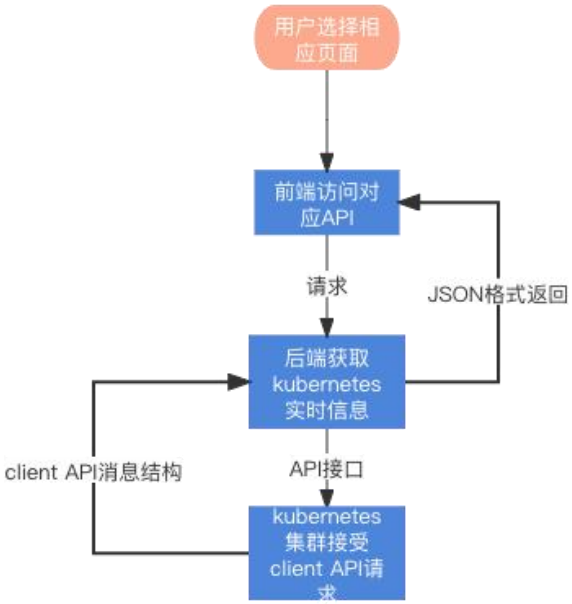

图 7 集群基本信息获取流程图

由于在 kubernetes 中，包括运行的 Pod，Node，用于部署的 Deployment，Replication Set 以及向外暴露接口的 Service 在内的每一个配置或实例，都是以 RestFul(Representational state transfer，表现层状态转换)设计中的 Representation(表现层)形式保存在分布式数据库 ETCD 中的[27]，获取到这些表现层的信息即获取到了相应 Pods，Services，Deployments 等配置或实例实时运行的状态信息。

并且根据 kubernetes 对表现层的设计，一般的表现层具有四个重要的键，即 Type Metadata(类型相关的元信息)，Object Metadata(对象相关的元信息)，Spec(描述)，Status(状态)。这四个重要的键中，Spec 对应集群操作员或控制器写入的希望对象呈现的属性或未来会呈现的属性，Status 对应对象当前的属性。例如，对于一个 Pod 类型的表现层来说，它的名称会记录在 Object Metadata 中，它的配置中对应的端口，挂载的镜像，文件系统，复制个数会显示在 Spec 中，它当前的状态，当前端口，运行个数会显示在 Status 中。Kubernetes worker node 上的 kubelet 会保证当前 status 尽可能与 spec 同步。

所以对于监控模块来说，只要掌握了 Spec，Status 和 Object Metadata 中的重要信息，就掌握了相应表现层的基本状态信息，配置信息和唯一的名称或 ID 信息。

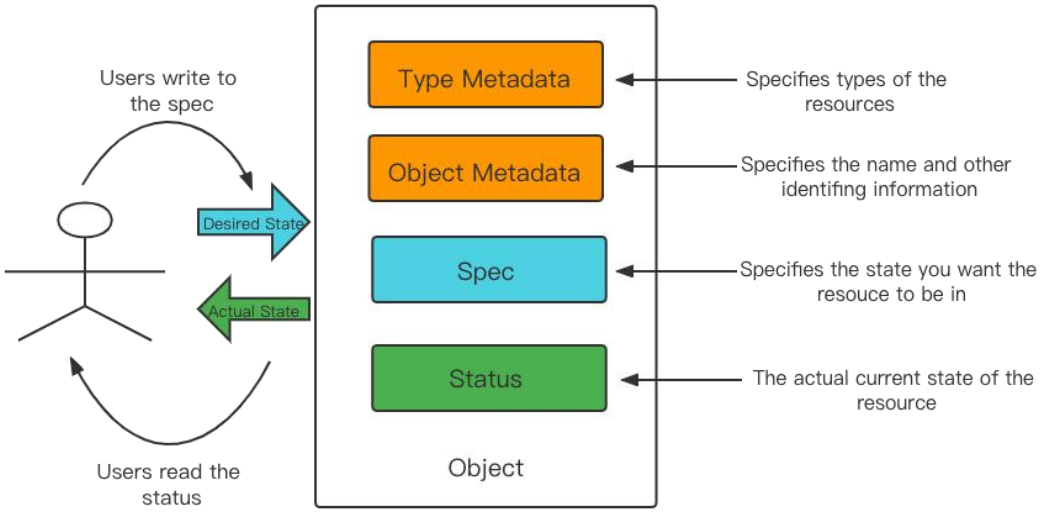

图 8 Kubernetes 中表现层键值对分类[27]

后端通过与 kubernetes client 交互后获得上述表现层信息，会将相应的重要信息解析出来，以 json 格式返回。其中，Pod 对象返回的格式如下：

```
{
  "label": {...},
  "monitored": false,
  "name": "buyservice-6978f578df-6svnq",
  "spec": {
    ...
    "containers": [...],
    ...
    "node_name": "vm-4c8g-node2",
    "node_selector": null,
    ...
    "volumes": [...],
  }
}
```

Deployment 对象返回的格式如下：

```
{
  "available_replicas": 1,
  "name": "buy-service",
  "replicas": 1,
  "template": {
    "metadata": {...},
    "spec": {...}
  }
}
```

Service 对象返回的格式如下：

```
{
  "name":"buy-service",
  "spec":{
    "cluster_ip":"xx.xx.xx.xx",
    ...
    "external_name":null,
    ...
    "load_balancer_ip":null,
    "ports":[...],
    ...
    "selectors":{...},
    ...
    "type":"ClusterIP"
  }
}
```

除了对集群基本运行和配置状态进行监控外，为了利于后续的分析预测模块，还需要对于目标 Pod 进行实时运行状态的监控。目标 Pod 的实时运行状态涉及到许多与 node 性能相关的指标，诸如 CPU 利用率，内存利用率，HTTP 请求增量等。这些指标并不在 kubernetes 的表现层中，需要搭建监控服务，时序数据库服务进行收集和存储。

在监控服务这一部分，监控模块使用 Prometheus 对整体集群进行监控。Prometheus 是用于事件监控和警报的开源软件应用程序，它将实时指标记录在使用 HTTP 拉取模型构建的时间序列数据库中，具有灵活的查询和实时警报功能[28]。许多现代化的云原生应用通过 Prometheus 结合 Grafana[29]搭建服务监控的大盘。

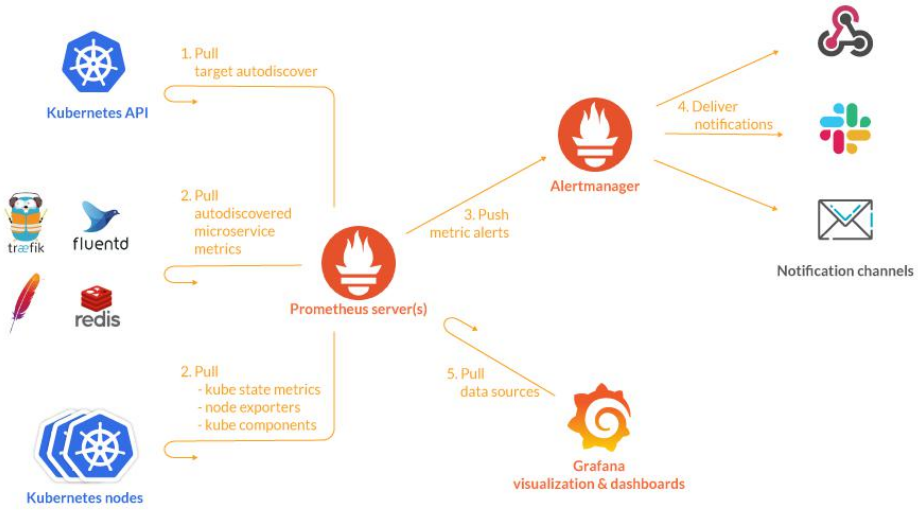

图 9 Prometheus 监控架构示意图

为了便于后续主动式部署策略的输入和执行，本工具将 Prometheus 部署模式设置为对整个集群进行监控。这样做有两点好处，一是针对后续加入的模拟负载，不需要进行额外的配置，提高了系统的可维护性和可扩展性；二是该设计适用于模拟云环境中 Pod 多次重启，迁移等操作导致的 IP，端口的变化。

在本工具中，Prometheus 的监控技术栈主要包含三个组件，分别是 cAdvisor，Kube-state-metrics 和 Metric-server。它们的具体作用和配置如下：

cAdvisor 是一个开源的容器资源利用率和性能分析代理。cAdvisor 内置在容器中并支持原生 Docker 的应用，作为 kubernetes 系统执行程序的一部分运行。因此，任何指标聚合器会在 cAdvisor 的作用下将容器指标作为 Prometheus 的终端点。

Kube-state-metrics 是一个响应 kubernetes API 并生成跟对象状态有关的指标的服务。这些指标包括 deployment，node，service 当前的状态。Prometheus 的指标聚合器会将 kube-state-metrics 当做一个指标终端点。

Metrics-server 是一个集群范围的资源利用率指标聚合器。但它只呈现最近的数据结果并不保存长期的数据。

成功搭建好 Prometheus 监控服务后可以通过访问服务暴露出的接口并输入相应指标的查询语句查询对应的指标。本工具中主要使用 Prometheus 聚合器收集的每个 Pod 每分钟实际利用的 cpu 占 Pod 要求的 cpu 份额的比例作为 cpu 利用率指标，每个 Pod 实际利用的内存占 Pod 最大限制的内存的比例作为内存利用率指标。

由于 Prometheus 的监控机制并不能保证长期数据的存储，且并非集群中所有 Pod 的实时监控信息都是需要的目标监控信息，本工具在前端提供给用户一个类似购物车的选择机制，用户可以在集群上正在运行的 Pods 中选择一批需要同时开始监控的 Pods，并选择对应的操作。后端根据用户选择的 Pods 启动监控子进程，子进程持续对这些目标 Pods 的对应指标进行查询，直到用户选择停止对目标 Pods 的监控。

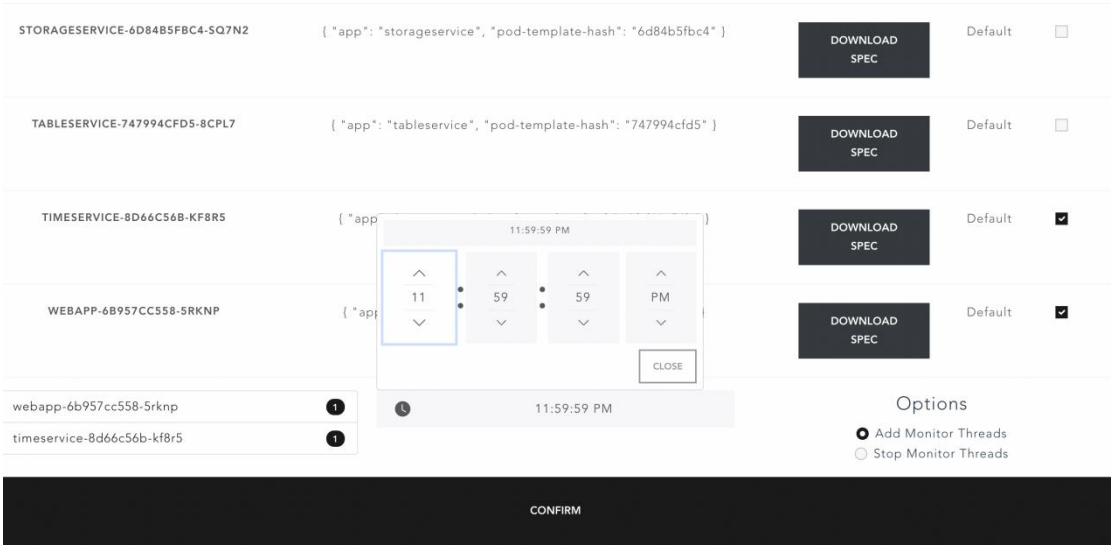

图 10 用户操作监控示例图

已经启动的子进程每隔 15s 访问 Prometheus 查询 API 查询一次对应 Pod 的指标，并清洗指标数据格式，将其存储在时序型数据库 InfluxDB[30]监控表里。子进程运行过程中，主

进程维护一个运行中的子进程表项，其中每一项记录着进程对应监控的 Pod 名称和进程 ID，用户不能重复开启已经处于监控中的进程。如用户终止监控进程，则主进程将该 Pod 对应的监控子进程杀死，并从子进程表项中移除。整体流程图如下所示：

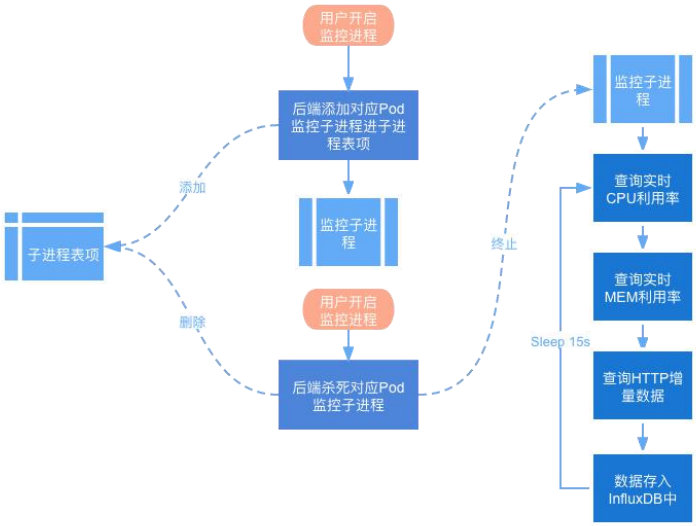

图 11 监控子进程启动/终止流程图

在监控中的 Pod 其指标变化可以在可视化模块中动态展示，该部分内容将在 2.1.5 小节中详述。

2.1.4 分析预测模块

分析预测模块需要实现根据用户的需求在模型池中抓取对应的模型，提取相应 Pod 的输入指标，并将输入指标清洗为模型输入对应的格式，获取预测的模型输出后反馈到前端，并在线更新模型。整体流程如下图所示：

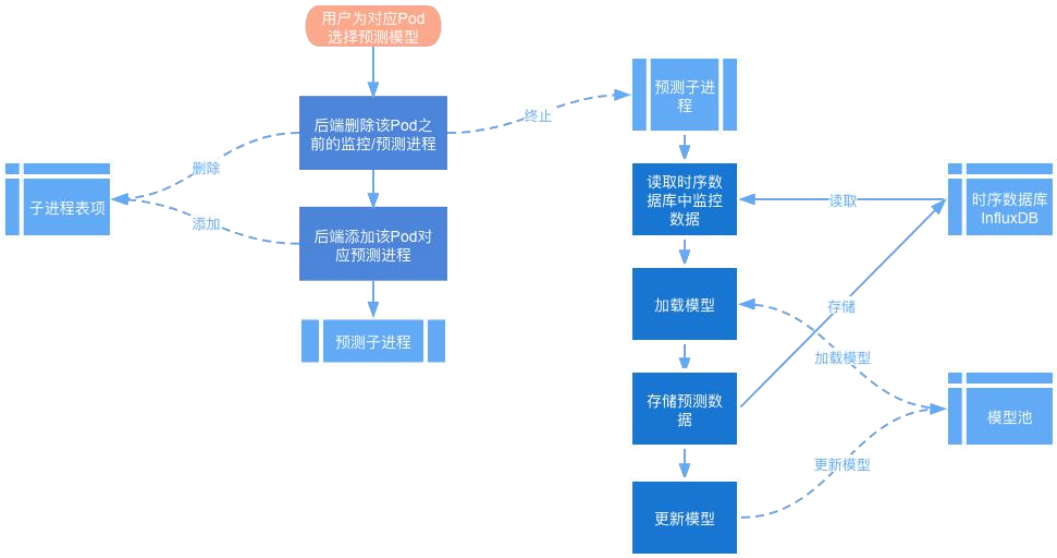

图 12 分析预测流程图

在上述流程中，主进程维护一个运行中的子进程表项，需要注意的是该表项不同于前一章节中的监控模块子进程表项，该表项存储的是 Pod 名称，预测模型名称和对应的预测子进程 ID。前一章节的监控模块子进程主要负责查询相应指标存入时序数据库监控表中，本章节的预测模块子进程主要负责从对应 Pod 的监控表中读取监控数据，并将该数据作为输入传入对应模型中，输出的数据则存入时序数据库预测表中。

该子进程表项保证了正在运行的预测子进程的记录，保证用户可以及时终止预测子进程的运行，并对于同一个 Pod 可以同时开启多个预测子进程。

预测子进程每次循环都会去检查数据库对应监控表中是否存在新的数据（时间戳更新的数据），如果不存在新的数据则进程睡眠 5 秒重复循环，如果存在新的数据则按照相应模型设定的回溯窗口切分相应长度的数据作为模型输入数据。

分析预测相关模型的设计与部署在 2.2 章节中详述。

预测的 Pod 指标变化和模型的评估指标可以在可视化模块中动态展示，该部分内容将在 2.1.5 小节中详述。

2.1.5 可视化模块

可视化模块主要基于 Vue 框架搭建，通过解析挂载后端传入的相应数据动态直观地展示集群的基本信息和 Pod 的实时运行情况以及预测值。

在基本信息的可视化这一块，可视化模块会对访问的端口返回的 json 格式数据进行解析，以表格的形式展示相应的数据。

集群上部署的 deployments 在展示的时候会显示名称，复制个数，正在运行中的复制个数，并提供下载 spec 文件的按钮。

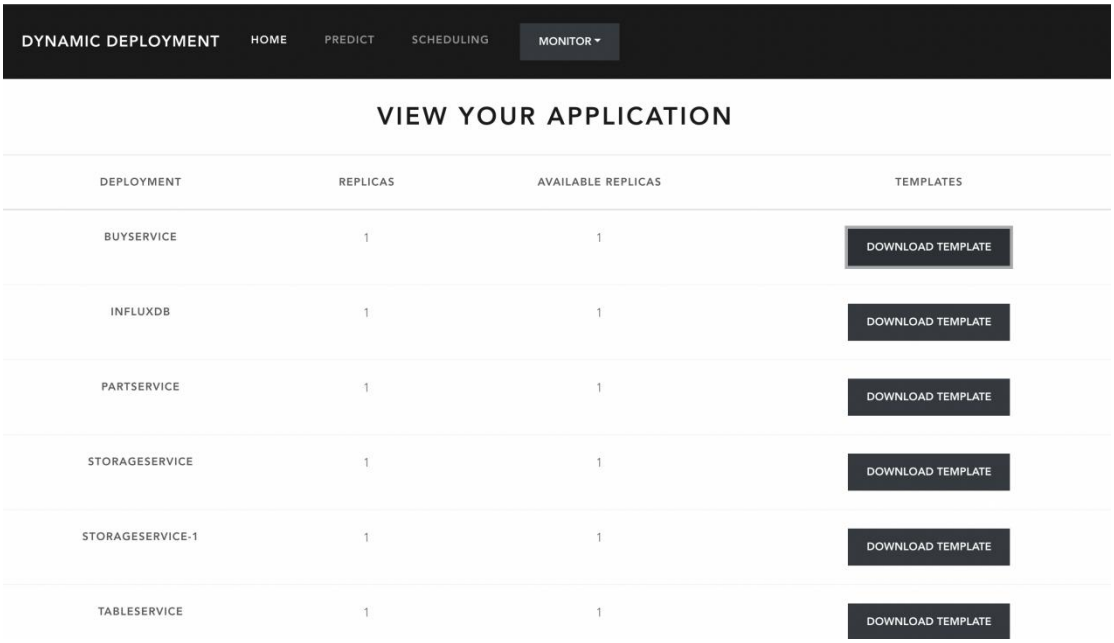

图 13 Deployments 可视化示例

集群上部署的 services 在展示的时候会显示名称并提供下载 spec 文件的按钮。

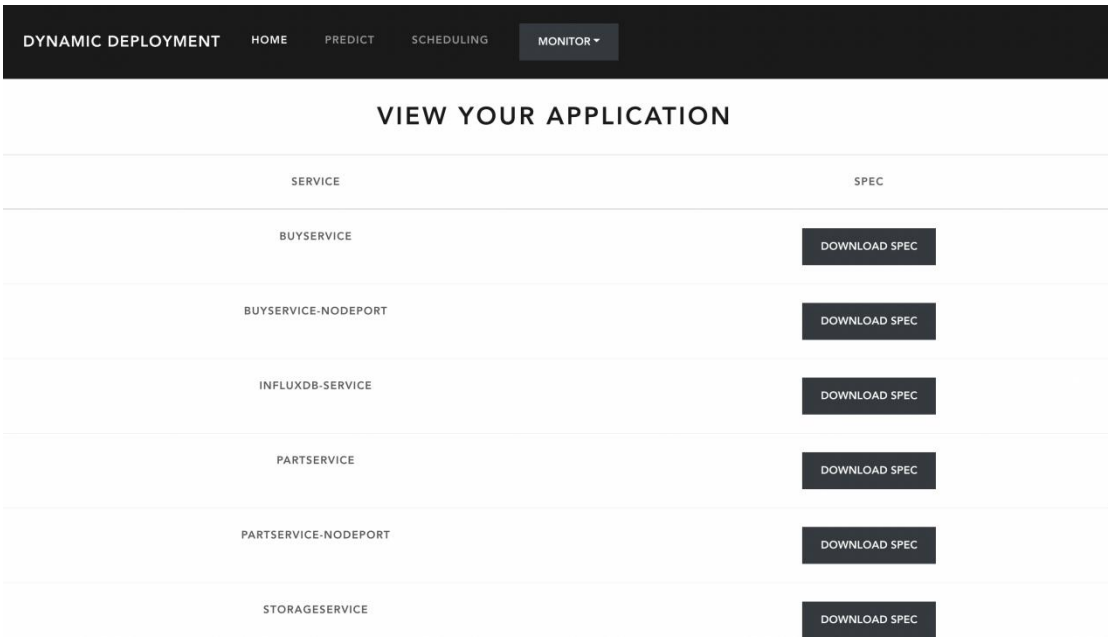

图 14 Services 可视化示例

集群上正在运行的 Pod 在展示的时候会显示名称，label，状态，提供下载 spec 文件的按钮。其中，状态这一栏是提示用户当前的 Pod 是否已开启监控子进程，default 状态代表未开启监控的状态，monitoring 状态代表已开启监控的状态。

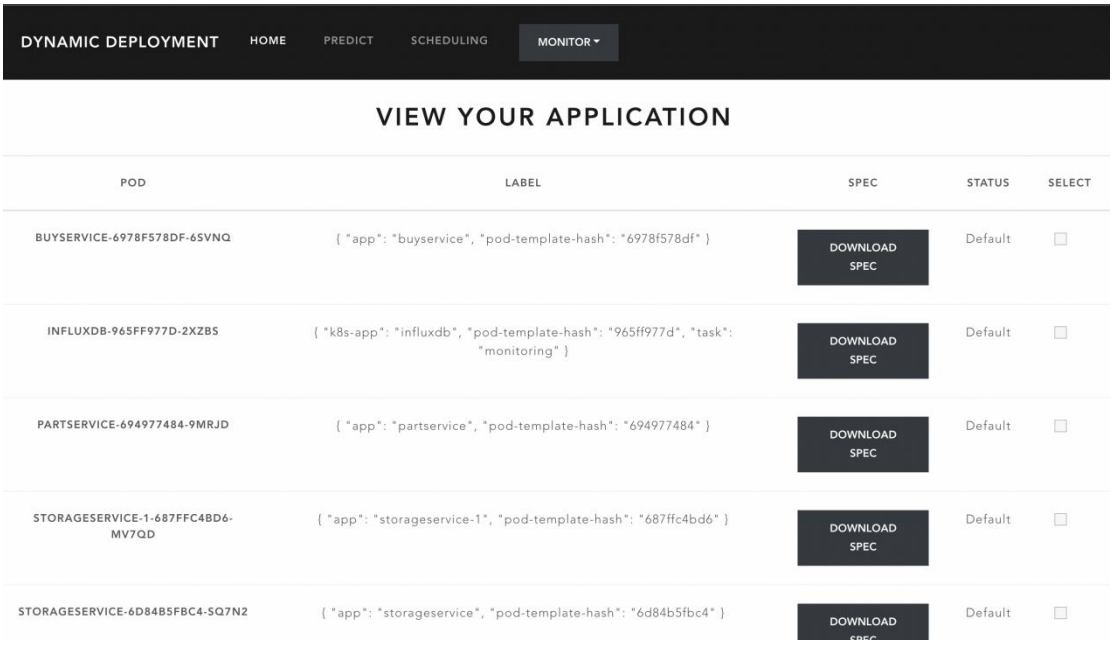

图 15 Pods 可视化示例

关于具体监控信息的可视化这一块，前后端通讯的设计采用了 websocket 协议。因为 websocket 协议不同于 HTTP 协议，其设计模式是基于客户端和服务端有双向通讯的情景。在分析预测模块的场景中，服务端需要将更新的监控数据及时推送到客户端。同时，由于在

客户端可能出现多个 Pod 监控可视化页面同时打开的情况，或者多个用户同时监控同一个 Pod 的情况，更新的监控数据需要保证推送到正确的接收端。

因此，在具体监控信息的可视化这一块，主要采用了类似聊天室架构的设计，在前端打开页面的瞬间前后端建立特定名称的长连接，长连接建立后前端申请订阅特定名称空间下的消息(在监控这一部分，该特定名称空间为{Pod 名称}-monitoring)，同时后端启动特定名称空间下的推送数据子进程，每隔 15s 检查对应数据库表中是否有新数据（时间戳更新的数据），如有新数据则推送到前端。前端页面关闭前订阅取消，连接关闭，同时后端终止推送数据进程。

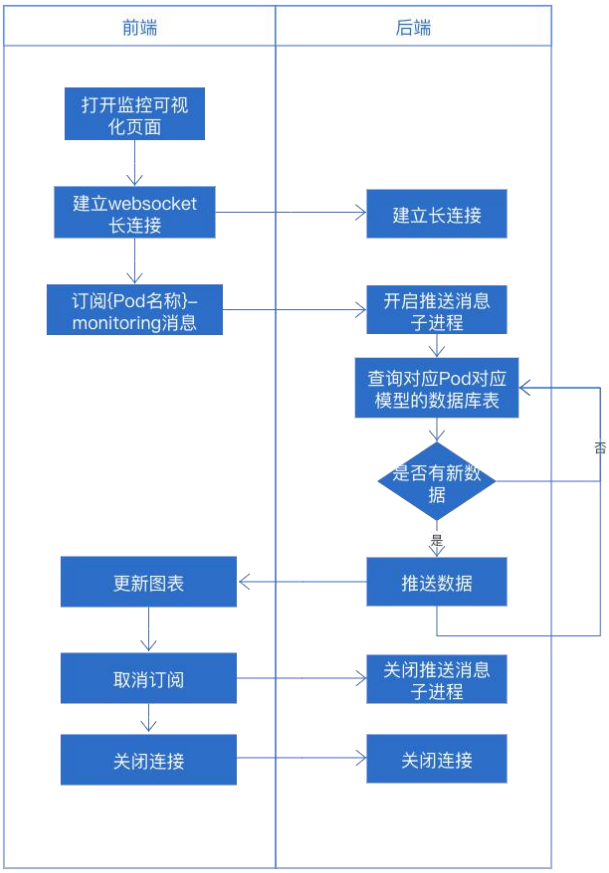

图 16 监控具体指标消息推送流程图

这一部分主要在前端使用了 Vue-socketio，socketio 包，在后端使用了 flask-socket 第三方库并配置了相应的允许跨域请求列表。

在前端图表的显示上，主要使用了基于 TypeScript 开发的 Echarts，通过动态绑定图表数据，响应后端推送数据的数据处理函数来更新数据实现图表的动态更新。

用户可以在该监控页面中选择模型池中的预测模型并点击 switch 按钮，前端会取消对之前消息的订阅并订阅新的名称空间为{Pod 名称}-{预测模型名称}的消息，后端接收到该消息会终止之前的推送消息子进程，开启新的名称空间对应的推送消息子进程。

2.2 基于微服务相关性的负载预测算法的设计与部署

2.2.1 方案设计背景

在微服务容器化的应用部署背景下，由于微服务应用的分布式架构（DAG 图结构）使得容器上运行的负载具有了显著的相关性关系，主要体现在两个方面：

1. 基于微服务之间调用或事件触发关系的不同微服务之间的负载相关性
2. 基于负载均衡技术产生的相同微服务不同实例之间的负载相关性

因此，如果能够充分挖掘微服务负载相关性将有助于负载变化的分析与预测，同时使模型具有良好的泛化性能和可解释性。

2.2.2 现有工作不足

现有工作大多没有考虑不同微服务(容器或虚拟机)负载之间的空间相关性，仅关注负载自身的时间相关性关系，由于云环境的动态性，微服务自身负载中存在大量的噪声，仅基于自身负载数据的模型容易受到数据噪声的影响，模型可靠性和稳定性不足。

目前已有部分工作关注不同微服务之间的空间相关性来辅助负载预测，它们采用的方法一般是如下两种：

1. 基于机器学习或深度学习模型从所有微服务历史负载数据中挖掘时空相关性，即直接将所有历史负载数据输入到神经网络中进行学习，其中的空间相关性不可解释，且模型本身容易受到相关性低甚至无关的微服务负载数据的影响。
2. 基于原始负载序列计算相关性（诸如欧氏距离，互相关，皮尔逊相关系数，聚类等）以寻找具有相关性的负载序列，建立统一的预测模型或者建立集成模型。

上述的这两种方法有以下几点不足：

1. 要求不同微服务负载序列数据上高度相关的假设是困难的；不同微服务之间可能出现趋势相关、周期相关、数据相关等不同特点，仅利用数据相关使得相关性信息遗漏或相关负载难以捕捉，从而造成模型生成困难或模型可靠性不足的问题。
2. 模型相关性建模粒度粗糙：没有提出统一模型以细粒度且全面地方式分析负载之间的多种相关性，模型在不同场景下的建模能力不足。
3. 模型开销较大：通常在获得大量数据的前提下离线训练模型，模型训练开销大且难以自适应负载的相关性变化。
4. 与资源管理结合不足：负载预测模型没有充分利用负载相关性关系辅助资源管理。

2.2.3 基于微服务相关性的负载预测算法设计

为全面利用微服务负载的相关性，基于微服务相关性的负载预测算法设计主要包括分析并建模微服务实例负载间的多样化相关性关系，包括：

1. **空间相关性**：不同微服务实例负载之间的相关性
2. **时间相关性**：微服务实例自身负载时间上的相关性
3. **特征相关性**：微服务实例负载多个子特征序列之间的相关性
4. **多尺度负载相关性**：远期负载、中期负载、近期负载与未来负载之间的相关性
5. **异常事件相关性**：微服务负载的变化具有突变性，这种突变性大多由于外部事件引发。如：节假日、自然灾害等特殊事件

其中，特征相关性的获取主要以时间序列分解技术作为基本技术，从微服务负载原始时间序列数据分解出趋势特征，周期特征和波动特征这三种特征，结合微服务负载原始时间序列数据，在这四类数据的基础上挖掘序列间的数据相关性，趋势相关性，周期相关性，波动

相关性及他们之间的大小，时间和正负关系。

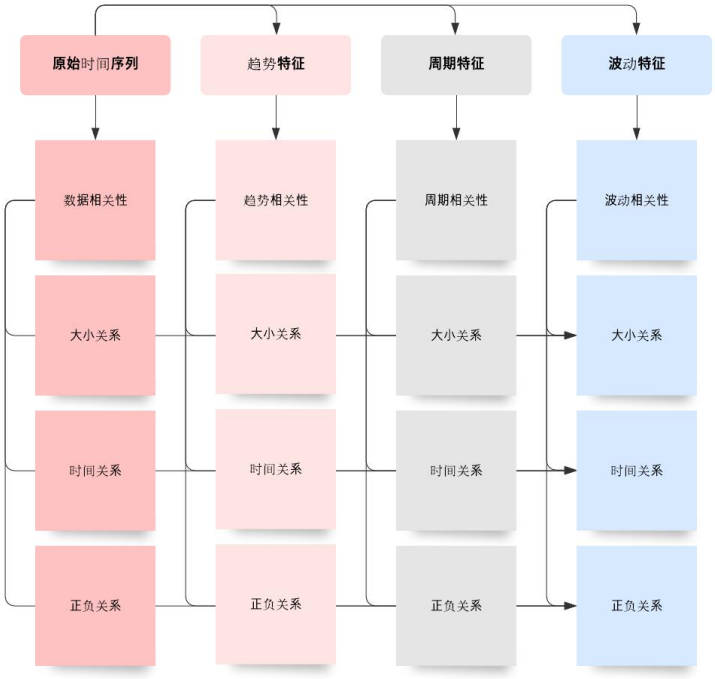

图 17 特征相关性关系挖掘

多尺度负载相关性主要是在微服务运行的时间轴上，考虑多个尺度的负载与未来负载之间的相关性。

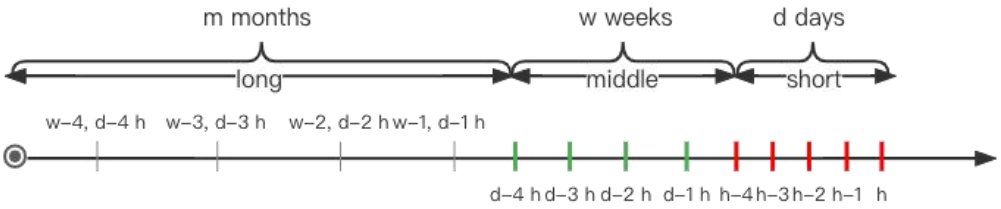

图 18 多尺度负载相关性示意图

接下来需要构建深度学习模型全面捕捉负载之间的相关性，以建模负载相关性特征与未来负载之间的复杂关系。具体步骤如下：

获取原始负载序列：假设历史负载中有 5 个容器的负载序列，其中待预测容器为 a。

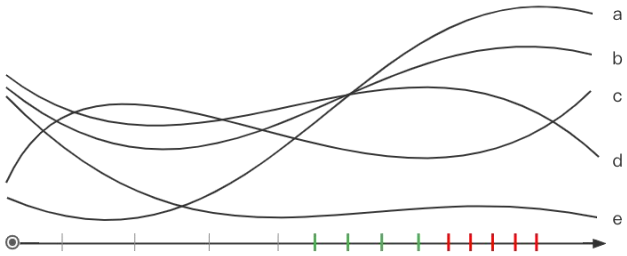

图 19 候选原始负载序列示意图

获取四种子特征负载序列：接下来对于候选微服务负载原始数据序列中的每个序列都进行时间序列分解，提取出其数据特征 DS，趋势特征 TS，周期特征 PS，波动特征 FS；这些特征构成一组候选特征序列。

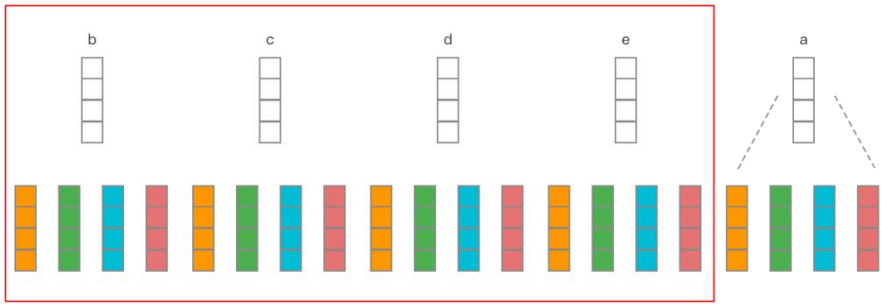

图 20 时间序列分解示意图

特征分组与对齐：针对每一种特征子序列，通过互相关公式计算目标微服务与任意微服务对应特征序列之间的相关性。通过互相关公式可以知道两个子序列之间的相关性程度（大小），相关性关系的时间顺序和相关性关系的正负性。其中，获得相关性大小是为了选出与目标微服务子特征最相关的负载序列，获得相关性时间顺序和相关性正负性是为了在特征拼接的时候进行特征对齐。

$$ccf_n = f(X_t, X_{t+n}) = r_{X_t X_{t+n}} = \frac{\sum(X_t - \bar{X_t})(Y_{t+n} - \bar{Y_{t+n}})}{\sqrt{\sum(X_t - \bar{X_t})^2 (Y_{t+n} - \bar{Y_{t+n}})^2}}$$

计算出相关性程度后，针对每种特征类别，选出最相关的 Top K 个负载特征序列。

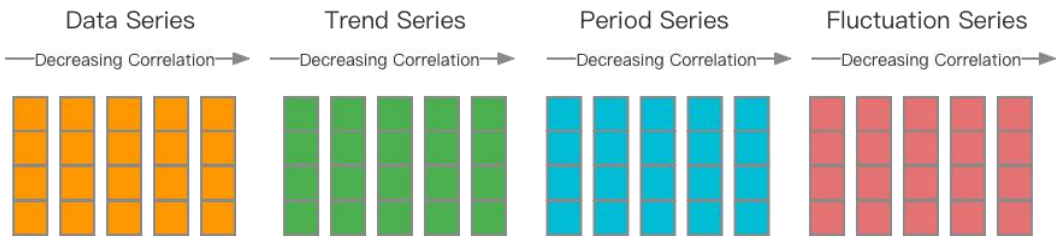

图 21 特征分组与对齐示意图

构造深度学习模型的输入特征：上述步骤获得的负载特征序列将作为深度学习模型的输入，深度学习模型输入神经元第一个通道代表目标微服务的四个子特征序列，第 2、3、4 通道分别代表与目标微服务对应子特征序列最相关的 Top1、2、3 的子特征负载序列。

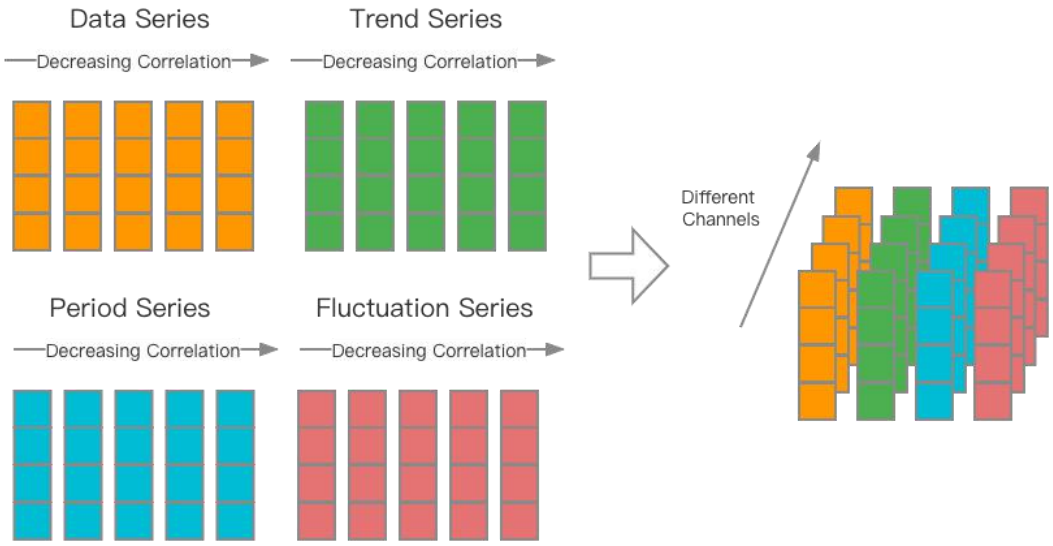

图 22 构造深度学习模型的输入特征示意图

提取空间相关性：提取空间相关性即提取不同微服务相同子特征序列的相关性。如下图所示，深度学习模型通过对输入特征图中的相应部分进行卷积计算一步步提取特征抽象，得到最后的 1 通道特征图。最后的 1 通道特征图中的每个值都来自于原始的不同微服务相同时刻相同子特征序列的加权和，从而达到了提取空间相关性的目的。

图 23 提取空间相关性示意图

装 订 线

提取时间相关性：提取时间相关性即提取时间维度上子特征序列自身的相关性。该步主要以历史四个时刻的不同子特征序列作为输入，未来四个时刻的不同子特征序列作为输出，通过时序神经网络(GRU[31])建模时间相关性。

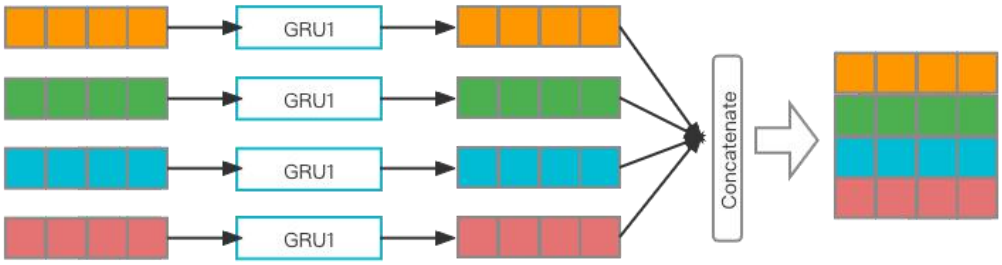

图 24 提取时间相关性示意图

提取子特征相关性：提取子特征相关性即提取任意两个子特征之间，任意三个子特征之间，任意四个子特征之间的相关性。由于直接对输入特征图卷积会是的特征降纬，为了保证特征尺寸不变，需要通过 padding[32]使得以上三种卷积的结果特征图与输入特征图尺寸一致。另外，由于卷积只能提取相邻特征之间的相关性，在固定的子特征序列排序顺序下，卷积无法提取跨行的子特征序列之间的相关性。因此，本算法在构造训练样本时采用随机采样的方式，每个样本的四个子特征序列的顺序随机生成。只要训练样本充足，模型就既能够学习到不同子特征序列之间的相关性，又能够避免排列组合所带来的模型高复杂度。

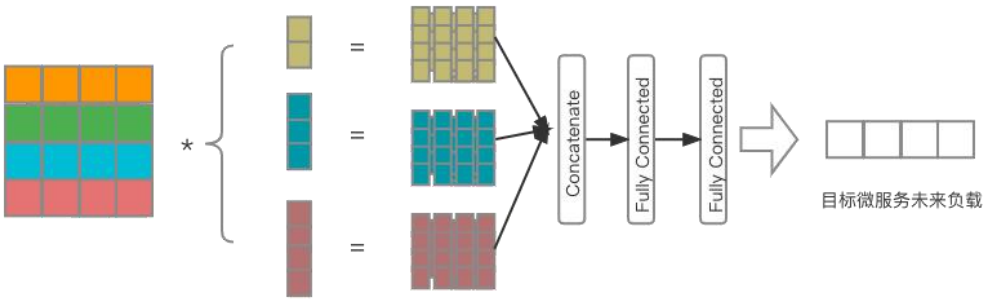

图 25 提取子特征序列相关性示意图

提取多尺度时间序列相关性：提取多尺度时间序列相关性即提取在远期，中期，近期的子时间序列与未来的序列之间的相关性。其中，短期序列即距离预测点最近的 m 个时刻；中期序列即增大一级周期粒度，距离预测点最近的 m 个时刻；长期序列即增大两级周期粒度，距离预测点最近的 m 个时刻。

提取异常事件相关性：异常事件向量化后作为模型的额外输入。

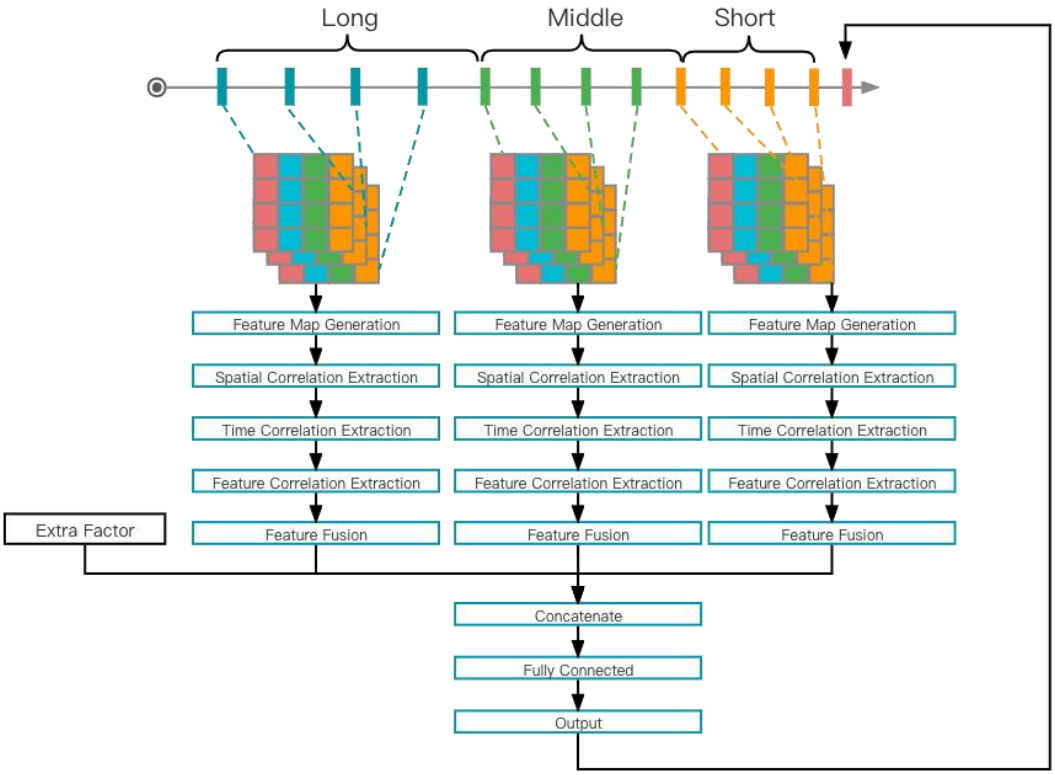

图 26 整体模型架构示意图

2.2.4 模型部署与在线更新

模型采用离线训练的方式，收集云平台上的历史负载数据，进行充分的分析训练。代理线训练完成后，模型将注册到平台的模型池中，并在线预测未来的微服务负载变化。为了适应微服务负载的新变化，模型进一步可以通过在线更新的方式，使用新负载数据增量式更新模型。模型部署并注册后，用户可以在可视化界面选择基于微服务相关性的负载预测模型并直观地看到模型预测的结果与真实值的偏差和与其他模型的比较。

3 实验分析

3.1 评价标准

3.1.1 负载预测工具评价标准

负载预测工具应具有完善的系统设计方案，使用文档；其功能应覆盖对 kubernetes 集群的整体监控，对 Pod 实时运行状态的监控，对监控数据的读取，对 Pod 未来运行状态的预测，同时具有简洁直观的可视化界面。

除此之外，为了实现后续毕业设计的要求，该负载预测工具应具有较高的可扩展性，模块之间耦合度低，可扩展、可重利用。

3.1.2 基于微服务相关性的负载预测算法评价标准

本实验采用均方误差(mean-square error, MSE)作为模型的评估指标。均方误差是指参数估计值与参数值之差的平方的期望，常用于回归预测任务的评估。使用均方误差作为模型的评估指标可以评估模型的预测值与真实值之间的变化程度。

3.2 实验结果与分析

3.2.1 负载预测工具的评估与分析

本文中的负载预测工具具有监控集群信息，监控 Pod 实时运行状态，存储 Pod 实时运行状态，查询 Pod 实时运行状态，预测 Pod 未来运行状态，可视化的功能。在功能范围上覆盖了 Kubernetes 常见监控与管理体系中 InfluxDB，Prometheus 和 Grafana 的基本功能，满足实验要求。

工具名称	监控集群基本信息	监控实时运行状态	存储运行状态	查询集群基本信息	查询实时运行状态	可视化
Kubernetes Client	√			√		
InfluxDB			√			
Prometheus	√	√		√	√	
Grafana						√
负载预测工具	√	√	√	√	√	√

表 1 kubernetes 体系监控与查询工具功能表

在模块的耦合设计上，各模块的代码分离作为 lib 目录下的子模块，在整体系统运行过程中作为自定义的 Python 模块引入。该负载预测工具具有功能上的高扩展性，新插入的模块可在单元测试完毕后作为自定义模块插入系统启动代码中。

在模型的扩展上，负载预测工具代码中使用的模型自模型池中引入，扩展的模型在模型池中注册后即可供业务代码使用。

在可视化界面上，通过类聊天室消息推送的模式实现了不同订阅名称的消息分离，用户可同时监控不同 Pod 的运行状态并使用不同的预测模型对 Pod 未来的运行状态进行预测。

3.2.2 基于微服务相关性的负载预测算法的结果与分析

在模型的评估阶段，采用四个基准算法：贝叶斯回归，线性回归，弹性网络，支持向量回归与本文提出的基于微服务相关性的负载预测方法相比较，在 Google Cluster Data 上使用过去的 1000 个观测点预测未来的 100 个观测点数据。

经比较得出，本文提出的基于微服务相关性的负载预测算法在均方误差这一指标上较贝叶斯回归模型减少 4%，较线性回归模型减少 3.9%，较弹性网络模型减少 7.7%，较支持向量回归模型减少 4.9%，整体性能优于上述四个基准模型。

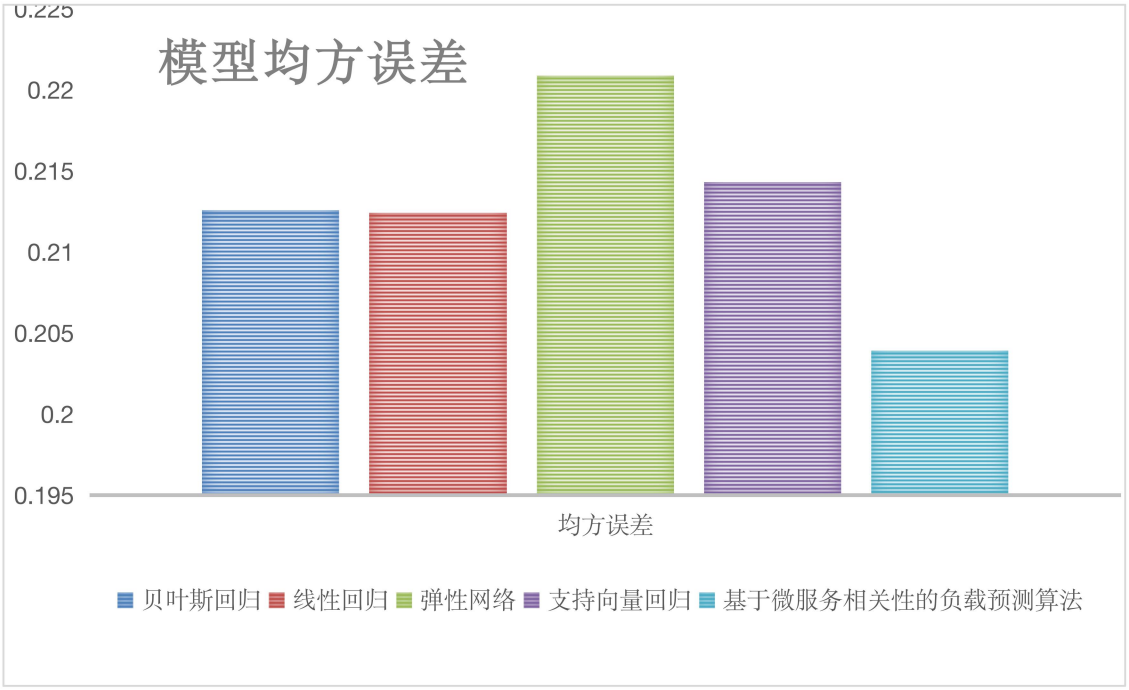

图 22 模型均方误差结果图

4 结论与展望

4.1 课题总结

本文首先介绍了在微服务容器化的应用部署背景下，微服务负载预测的意义和目的。接下来概括总结了目前微服务负载预测的研究现状和基本分类，针对部分算法，列举了其不足与可借鉴之处。

然后本文着重介绍了负载预测工具的设计与开发。本文中的负载预测工具主要基于亚马逊提出的云基础架构的自动式管理方案——MAPE 循环进行设计。文中详细介绍了负载预测工具的整体物理部署方案，具各个子模块的设计和工作流程，最后对负载预测工具进行了简要的评估。

在基于微服务相关性的负载预测算法部分，本文描述了基于微服务相关性的负载预测算法的设计背景，基本思路和模型架构以及模型如何与负载预测工具的分析预测模块相结合。最后，在 Google Cluster Data 数据集比较该模型与其他的基准模型，并列出了模型评估指标和相应的结果。

4.2 后续工作

在毕业设计阶段，我将针对现有的负载预测工具进行扩展设计与开发，加入部署决策模块和执行器模块；设计响应式部署模型和主动式部署模型并在模拟数据集上训练，验证两个模型，在模型验证完成后将主动式部署模型加入模型池并完成模型决策在 kubernetes 上的落地；针对主动式部署方案设计直观的可视化界面，在现有的可视化模块中加入适合主动式部署方案的可视化界面。

参考文献

[1] Masdari, M., Khoshnevis, A. A survey and classification of the workload forecasting methods in cloud computing. Cluster Comput 23, 2399 – 2424 (2020). https://doi.org/10.1007/s10586-019-03010-3

[2] Antonescu A F, Braun T. Simulation of SLA-based VM-scaling algorithms for cloud-distributed applications[J]. Future Generation computer systems, 2016, 54: 260-273.

[3] Yang J, Liu C, Shang Y, Mao Z, Chen J (2013) Workload predicting-based automatic scaling in service clouds. In Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on IEEE. p 810 – 815

[4] Tong, J.J., Hai-hong, E., Song, M.N., Song, J.D.: Host load prediction in cloud based on classification methods. J. China Univ. Posts Telecommun. 21(4), 40 – 46 (2014)

[5] Zhong, W., Zhuang, Y., Sun, J., Gu, J.: A load prediction model for cloud computing using PSO-based weighted wavelet support vector machine. Appl. Intell. 48(11), 4072 – 4083 (2018)

[6] Nikravesh AY, Ajila SA, Lung CH (2015) Towards an autonomic auto-scaling prediction system for cloud resource provisioning. In Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems.IEEE Press, p 35 – 45

[7] Cetinski, K., Juric, M.B.: AME-WPC: advanced model for efficient workload prediction in the cloud. J. Netw. Compu. Appl. 55, 191 – 201 (2015)

[8] Yang, Q., Zhou, Y., Yu, Y., Yuan, J., Xing, X., Du, S.: Multistep-ahead host load prediction using autoencoder and echo state networks in cloud computing. J. Supercomput. 71(8), 3037 – 3053 (2015)

[9] Patel, Y.S., Misra, R.: Performance comparison of deep VM workload prediction approaches for cloud. In Progress in Computing, Analytics and Networking, pp. 149 – 160. Springer, Singapore (2018)

[10] Gupta S, Dinesh DA (2017) Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks. In 2017 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS). IEEE, p 1 – 6

[11] Pacheco-Sanchez S, Casale G, Scotney B, McClean S, Parr G, Dawson S (2011) Markovian workload characterization for qos prediction in the cloud. In 2011 IEEE 4th International Conference on Cloud Computing. p 147 – 154

[12] Jheng JJ, Tseng FH, Chao HC, Chou LD (2014) A novel VM workload prediction using Grey Forecasting model in cloud data center. In The International Conference on Information Networking 2014 (ICOIN2014). p 40 – 45

[13] Kluge F, Uhrig S, Mische J, Satzger B, Ungerer T (2010) Dynamic workload prediction for soft real-time applications. In 2010 10th IEEE International Conference on Computer and Information Technology. p 1841 – 1848

[14] Ardagna D, Casolari S, Panicucci B (2011) Flexible distributed capacity allocation and load redirect algorithms for cloud systems. In 2011 IEEE 4th International Conference on Cloud Computing. p 163 – 170

[15] Qazi K, Li Y, Sohn A (2013) PoWER: prediction of workload for energy efficient relocation

of virtual machines. In Proceedings of the 4th annual Symposium on Cloud Computing, 2013: ACM, p. 31

[16] Hu Y, Deng B, Peng F, Wang D (2016) Workload prediction for cloud computing elasticity mechanism. In 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). p 244 – 249

[17] Liu, Y., Gong, B., Xing, C., Jian, Y.: A virtual machine migration strategy based on time series workload prediction using cloud model. Math. Probl. Eng. 2014, 11 (2014)

[18] Duggan J, Chi Y, Hacigu ¨ mu ¨ s¸ H, Zhu S, Cetintemel U (2013) Packing light: portable workload performance prediction for the cloud. In 2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW). p 258 – 265

[19] Zhang L, Zhang Y, Jamshidi P, Xu L, Pahl C (2014) Workload patterns for quality-driven dynamic cloud service configuration and auto-scaling. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, 2014: IEEE Computer Society. p 156 – 165

[20] Cao, J., Fu, J., Li, M., Chen, J.: CPU load prediction for cloud environment based on a dynamic ensemble model. Software 44(7), 793 – 804 (2014)

[21] Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). Google cluster-usage traces: format+ schema. Google Inc., White Paper, 1-14.

[22] Martin F. Arlitt and Carey L. Williamson. 1995. A workload characterization study of Internet Web servers. SIGAPP Appl. Comput. Rev. 3, 2 (Fall 1995), 1 – 4. DOI:https://doi.org/10.1145/228228.228229

[23] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," in IEEE Cloud Computing, vol. 1, no. 3, pp. 81-84, Sept. 2014, doi: 10.1109/MCC.2014.51.

[24] M. Maurer, I. Breskovic, V. C. Emeakaroha and I. Brandic, "Revealing the MAPE loop for the autonomic management of Cloud infrastructures," 2011 IEEE Symposium on Computers and Communications (ISCC), 2011, pp. 147-152, doi: 10.1109/ISCC.2011.5984008.

[25] Aggarwal, V., Halepovic, E., Pang, J., Venkataraman, S., & Yan, H. (2014, February). Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements. In Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (pp. 1-6).

[26] Kubernetes-client, https://github.com/kubernetes-client/python/tree/master/kubernetes

[27] LUKŠA, M. (2018). Kubernetes in action. http://proquest.safaribooksonline.com/9781617293726.

[28] Wikipedia contributors. (2022, January 3). Prometheus (software). In Wikipedia, The Free Encyclopedia. Retrieved 07:39, January 4, 2022, from https://en.wikipedia.org/w/index.php?title=Prometheus_(software)&oldid=1063477261

[29] Chakraborty, M., & Kundan, A. P. (2021). Grafana. In Monitoring Cloud-Native Applications (pp. 187-240). Apress, Berkeley, CA.

[30] Naqvi, S. N. Z., Yfantidou, S., & Zimányi, E. (2017). Time series databases and influxdb. Studienarbeit, Université Libre de Bruxelles, 12.

[31] Dey, R., & Salem, F. M. (2017, August). Gate-variants of gated recurrent unit (GRU) neural networks. In 2017 IEEE 60th international midwest symposium on circuits and systems

(MWSCAS) (pp. 1597-1600). IEEE.
[32] Liu, G., Shih, K. J., Wang, T. C., Reda, F. A., Sapra, K., Yu, Z., ... & Catanzaro, B. (2018). Partial convolution based padding. arXiv preprint arXiv:1811.11718.