使用时间卷积网络管理无服务器云中的冷启动

Tam N. Nguyen
tom.nguyen@ieee.org
linkedin.com/in/tamcs

arXiv:2304.00396v1 [cs.DC] 1 Apr 2023

摘要——**无服务器云是一种创新的云服务模型，使客户免于承担大部分云管理职责。它还具备其他云模型的诸多优势，而成本却低得多。因此，无服务器云日益广泛地应用于系统安全、银行业和医疗保健等影响重大的领域。冷启动是影响无服务器云性能的一大威胁：为满足客户请求而配置所需云资源所耗费的时间，会给服务提供商和／或客户带来无法接受的成本。本文提出了一种新颖的低耦合、高内聚集成策略，用于解决无服务器云技术栈基础设施级和函数级的冷启动问题，而当前最先进的策略关注范围则较为狭窄。该集成策略以预测未来 10 至 15 分钟内的函数实例到达情况为基础。通过采用时间卷积网络（TCN）深度学习方法即可实现这一目标。基于某大型无服务器云服务提供商真实世界数据集的基准测试结果表明，在时间序列预测方面，TCN 的表现优于其他常用机器学习算法。除了冷启动管理之外，所提出的策略及公开可用代码还可用于解决其他云问题，例如优化虚拟软件定义网络资产的配置。**

索引词—**冷启动、FaaS、时间卷积网络、云计算、深度学习**

I. 引言

云计算日益普及，因为它能够提升灵活性、效率和创新这三个战略业务目标。企业可以根据需求扩展其云基础设施，享受安全的预构建工具和高度可靠的存储选项，同时能够将某些职责转交给云服务提供商（CSP）。云计算还具有高效性，因为它提供了更广泛的可访问性、更快的上市速度、更一致的数据安全性、更低的设备成本，以及最重要的，更加灵活的付费结构。因此，企业能够更快地实现创新并将其创新成果商业化。

管理云系统涉及负载均衡、自动扩缩容、网络安全、监控等方面的挑战 [1]。为免去客户管理云服务的职责，Amazon 于 2014 年率先推出 Amazon Lambda，随后 Google 和 Microsoft 等大型云服务提供商于 2016 年也采用了这项服务。Amazon Lambda 是“无服务器云服务”或“函数即服务”（FaaS）的一个示例，它采用按需付费模式，并提供服务自动扩缩容功能 [2]。客户只需向云服务提供商注册其函数。收到调用后，云服务资源便会被分配，以成功执行这些函数。客户将收到执行结果，而无需操心幕后繁杂的云服务管理工作。

与其他云服务模型相比，无服务器模型得益于资源复用和基础设施异构性，因而更加经济实惠 [2]。例如，Deloitte [3]报告称，Amazon 无服务器服务具备与 Amazon EC2 相同的功能，但成本仅为其 65%。凭借这些出色的特性，无服务器云已越来越多地应用于高影响力领域，例如实时协作与分析、城市与工业管理、系统与软件安全、电子商务、医疗保健、银行业和区块链。因此，维持无服务器云的高性能水平对于云服务提供商（CSP）及其客户而言变得越来越重要。

冷启动是无服务器云性能面临的一大威胁。当为处理客户请求而配置所需云资源所花费的时间导致服务提供商和/或客户承担无法接受的成本时，就会发生冷启动。解决这一问题并非易事，因为这意味着必须始终保证资源可用。最主要的原因是，必须取消配置闲置资源，以控制成本。

通过文献综述，本文首先指出了冷启动管理中的五项现实关键挑战，以及应对这些挑战方面目前存在的不足（第二节）。为借助机器学习支持当前及未来的冷启动管理工作，本文将这五项挑战转化为两组具体的数据需求（DR）和预测模型需求（MR）。本文基于 Microsoft 的 Azure Functions Trace 2019 数据集 [1]，生成了满足所有 DR 的训练数据集（第三节）。第四节简要讨论了时间序列预测的主流算法，为选择时间卷积网络作为满足所有 MR 的算法奠定基础。该深度学习流水线由两个串联的模块组成，用于预测未来实例的函数名称（模块A），以及预测这些实例的到达时间（模块 B）。

基于能够预测未来 10 至 15 分钟函数实例到达情况的能力（第 V 节），本文提出了一种新颖的低耦合、高内聚集成策略，旨在解决无服务器云技术栈中基础设施级和函数级的冷启动问题（第 VI 节）。除了冷启动管理之外，所提出的策略及本文公开共享的源代码还可用于解决其他云计算问题，例如优化虚拟软件定义网络资产的配置。

表 I FaaS 工作负载时间预测的近期研究成果

研究成果	Year	预测目标	方法	C1	C2	C3	C4	C5
Mahmoudi 等人 [4]	2022	冷启动概率	半马尔可夫过程			●	●	●
Ebrahimpour 等人 [5]	2022	函数合并时间	依赖图上的启发式方法		●	●		●
Jegannathan 等人 [6]	2022	工作负载到达时间	季节性自回归综合移动平均模型		●		●	●
Steinbach 等人 [7]	2022	函数调用	神经时序点过程		●	●	●	●
Singhvi 等人 [8]	2021	工作负载到达率	指数加权移动平均			●	●	●
Lin 等人 [9]	2021	工作流响应时间	关键路径贪心算法		●	●		●
Shahrad 等人 [1]	2020	函数工作负载	直方图分析和 ARIMA		●	●		●
Anirban Das 等人 [10]	2020	端到端延迟	梯度提升回归器				●	●

[a]C1 至 C5 是前文提到的解决冷启动问题时面临的现实挑战。

II. 背景

无服务器服务可以是 FaaS，也可以是后端即服务（BaaS）[2]。本文聚焦于 FaaS，它由客户端维护的应用程序及其在 CSP 中已注册的函数组成。这些函数可以通过 HTTP 请求、定时器事件、命令行接口、推送到消息队列的更新或其他机制触发。收到触发请求后，API 网关会执行必要的检查，并根据客户端预先定义的规则，将触发请求映射到正确的函数。控制器与 API 网关协同工作，并决定计算池中的哪个工作节点接收哪个函数以执行。控制器通常由资源调度器、资源扩缩器和资源监控器三个主要组件组成 [11]。工作节点是一个精简、隔离的虚拟环境，仅包含成功执行函数所需的资源和配置。函数成功执行后，工作节点会将结果发送给控制器。随后，控制器将结果转发给 API 网关，再由 API 网关将结果返回给客户。控制器还可以关闭工作节点以节省资源。图 1 总结了 FaaS 工作流。

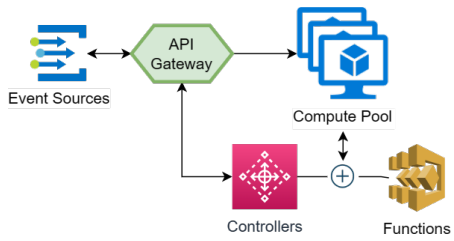


图 1. FaaS 工作流

值得注意的是，创建工作节点需要初始化新的容器、设置代码依赖关系、配置合适的运行时环境，以及满足其他要求 [12]。节点配置取决于自动扩缩策略，例如按请求扩缩、并发值扩缩或基于指标的扩缩 [4]。大多数云服务提供商采用按请求扩缩策略：当没有可用的已配置工作节点（即“预热节点”）来处理到达的函数请求时，便配置一个新的工作节点 [4]。当客户的无服务器函数请求的总执行时间由于新节点配置所需时间过长而长到无法接受时，就会出现冷启动问题。

通过使工作节点随时可用，可以解决冷启动问题。然而，这与会关闭节点的资源节约策略相冲突。大多数云服务提供商采用固定的保活策略，使工作节点保持固定时长 [1]。研究发现，该策略、延迟绑定策略和随机负载均衡策略效率不高，因为它们未能充分应对以下实际挑战 [1], [13]。

- 客户运行 FaaS 应用程序的需求难以预测。（**C1**）
- FaaS 应用程序具有异构性，其函数调用模式存在很大差异。（**C2**）
- 某些 FaaS 应用程序的运行频率非常低。（**C3**）
- FaaS 函数调用跟踪的成本应当合理。（**C4**）
- FaaS 函数优化策略的执行速度必须足够快。（**C5**）

有效的冷启动管理策略通常建立在充分开展工作负载特征分析、性能预测、资源调度和资源扩缩容的基础上 [11]。本文聚焦于预测 FaaS 工作负载这一任务，并将其作为资源调度和资源扩缩容的依据。表 I 列出了近期关于预测 FaaS 工作负载的研究成果。为明确主要区别，本文致力于解决准确预测未来 FaaS 函数实例到达时间的任务，并应对上述五项挑战；这些挑战具体体现为以下数据集和模型要求。

用于预测模型训练和验证的数据集必须：

- 源自主要 FaaS 云服务提供商的真实世界数据集。（**DR1**）
- 涵盖多样化的用户群体和应用程序。（**DR2**）
- 涵盖异构应用程序，且其调用难以预测。（**DR3**）
- 涉及一些执行频率极低的函数。（**DR4**）
- 没有明显的长期趋势或季节性变化。（**DR5**）
- 具有复杂的时间依赖关系。（**DR6**）预测模型必须：

- 在预测未来函数即服务函数到达时间方面具有竞争力的准确性。（**MR1**）
- 支持增量学习。（**MR2**）
- 基于过去步骤积累的泛化知识，支持预测未来适量的步骤。（**MR3**）
- 速度足够快。（**MR4**）

接下来的章节将进一步介绍数据集的构建和模型设计。

III. 数据集

本文使用了 Sharad 等人最初发布的 Azure Functions Trace 2019 数据集[1]。数据采集自真实世界的 Microsoft Azure 函数即服务（FaaS）环境（满足 DR1 和 DR2）。数据集及其完整文档可在 *github.com/Azure/AzurePublicDataset*查阅。最相关的数据字段如下。HashOwner 包含应用所有者的唯一 ID。HashApp 包含应用名称的唯一 ID。HashFunction 包含应用内函数名称的唯一 ID。Average 表示函数实例的平均执行时间。

每个函数实例的函数调用记录总时长为 14 天。函数触发器分为 7 类：HTTP、事件、队列、计时器、编排、存储及其他。一个应用程序可能包含多个函数。一个函数可以多次加载到不同的工作节点中。每个加载后的实例都有自己的调用计划。同一个函数可能存在多个实例，原因有很多，例如同一所有者编写的应用程序内部或跨应用程序的多次函数调用。图 2 展示了函数实例的到达情况及其调用计划示例。

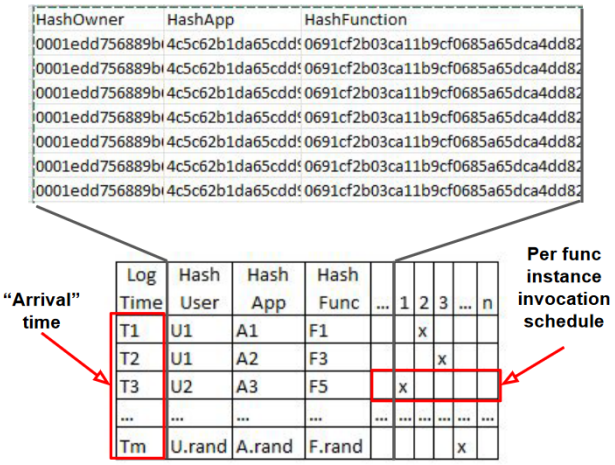


图 2. 函数实例到达时间与函数实例调用计划

在此示例中，同一应用、同一用户的同一函数共有 7 个实例。进一步检查数据集后，可以看到以下调用计划：实例 1 [578], 实例 2 [639, 642, 1288, 1292], 实例 3 [622, 627], 实例 4 [772, 773,776], 实例 5 [559, 566], 实例 6 [714, 723, 1100, 1102,1109],和实例 7 [677, 679, 690,704,1098]。这些数字表示 14 天全局时间线中的分钟位置。需要注意的是，这些逐实例调用计划不同于函数实例到达时间。Sharad 等人[1]致力于通过优化函数实例调用计划来解决冷启动问题，而本文则侧重于通过预测函数实例到达时间来优化函数实例进程。

A. 数据探索结果

数据集探索结果（包括详细图表）可参见 Shahrad 等人撰写的原论文 [1]。主要结果总结如下。

大多数函数的执行时间不超过初始化函数执行所需资源的时间。因此，冷启动确实是一个问题。绝大多数应用程序的调用频率较低，这符合 DR4。不到 20% 的应用程序占全部调用的 99.6%，这意味着不同应用程序之间的函数调用次数差异巨大。此外，不同应用程序中函数的平均到达间隔时间也存在很大差异。这两项发现符合 DR3 和 DR5。调用频率最低的函数，其执行次数比最热门的函数少 8 倍。这一发现支持 DR4。HTTP 触发器发挥着重要作用：64% 的应用程序至少有一个由 HTTP 触发的函数请求，43% 的应用程序只有由 HTTP 触发的函数 [1]。不同触发器的平均函数执行时间也存在很大差异。研究发现，内存是影响函数预热、分配和保持活动决策的重要因素。这两项发现支持 DR6。

根据数据探索结果，本文聚焦于以下目标。

1)**预测未来 HTTP 触发函数实例的名称**。2) **预测这些函数实例各自的到达时间**。

B. 数据准备

训练数据集使用 Azure Synapse Data Lake-House 和 PySpark 制备。原始数据集包含 662927 个 HTTP 触发函数实例。考虑到为期 14 天的整体时间跨度，每分钟约有 33 个已预配的 HTTP 触发函数实例。经过筛选并导出的数据集有 136497 行。主要数据字段包括函数索引（函数实例索引）、HashOwner、HashApp、函数 ID（函数实例名称的数字映射）和 Average（每个函数实例的平均执行时间）。包含每个数据处理步骤说明的数据准备 Jupyter 笔记本以及导出的数据集，可在*https://github.com/Cybonto/TCN4CLOUD*找到。关键的数据处理决策可简要概括如下。

仅选择了 HTTP 触发函数实例，因为与 HTTP 相关的调用最难预测，

也是原始 Azure FaaS 数据集中最常见的调用类型。随后，仅选择预配实例数超过 9 个的函数。每个函数实例都有自己的函数调用计划。采用这一筛选条件，并不会牺牲训练数据集中同时包含预配实例数少于 9 个和多于 9 个的函数所带来的复杂性。例如，沿时间线排列时，某些函数的较早实例（第 1 至第 5 个）仍会与其他函数较晚的实例（第 9 至第 15 个）混合出现。采用这一筛选条件的好处不仅在于获得高效的训练数据。只选择实例数超过 9 个的函数后，函数实例到达之间的时间间隔及其方差都增大了，从而更好地满足 DR3、DR5 和 DR6。事实上，按时间顺序排列的各行中，时间间隔的平均方差为 3.6 行。

IV. 模型设计

长度为 $n \in \mathbb{N}$ 的时间序列 T 可定义为数据点对的集合 $T = \{(t_i, p_i) | t_i \leq t_{i+1}, 0 \leq i \leq n\}$，其中数据点 $p_i \in R_d$ 具有 d 行为属性和时间戳 $t_i \in \mathbb{N}, i \leq n$。当 $d = 1, T$ 为单变量时，而当 $d > 1, T$ 为多变量时。图 3 展示了时间序列预测的关键概念。时间序列可能具有平稳性（均值、方差和自相关结构恒定）、季节性（周期性重复出现的事件）、趋势、脉冲等特征。根据 Azure ML 分析，训练数据集中既没有季节性，也没有明显趋势。因此，传统的时间序列机器学习算法并不适合实现这些目标。相反，本文着眼于已知性能更优的深度学习算法 [14]。

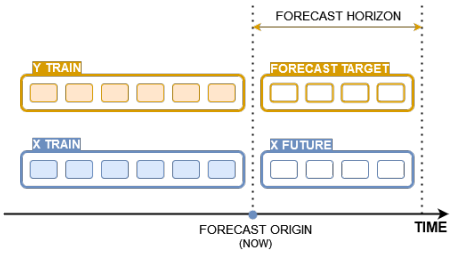


图 3. 时间序列预测

训练数据集可以看作由许多时间序列组成，每条时间序列对应某个函数实例的到达计划。为每条时间序列单独训练一个模型会违反 MR4，而且可能成本过高，无法满足 MR3。因此，深度学习模型必须是能够从所有时间序列中共同学习的全局模型。当该全局模型能够一次性准确预测未来数百步时，便能以最高效率满足 MR3 和 MR4。这个预测步数称为预测范围。具体而言，该模型的深度学习算法必须支持直接预测，而不是逐步预测或递归预测。例如，对未来第 2 步的预测不需要依赖对未来第 1 步的验证。这类模型可以形式化表示为：

$$P(z_{i,t+1:t+h} | Z_{1:t}, X_{1:t+h}; \theta_i), \theta_i = \Psi(z_{i,1:t}, X_{i,1:t+h}, \Phi),$$

我们将 $Z = \{z_{i,1:T_i}\}_{i=1}^{N}$ 定义为 N 单变量时间序列集合，其中$z_{i,\ 1:T_i}$ 为其取值。 $\{X_{i,1:T_i}\}_{i=1}^{N}$ 是包含时间特征和静态特征的协变量向量。 θ_i 是模型参数的 i^{th} 集合。 Φ 是所有 N 条时间序列共享的参数。 Ψ 是所选的深度学习算法。该模型可使用深度学习架构实现。

在 2022 年的一篇综述中，Benidis 等人 [14] 列出了以下用于时间预测的深度学习算法族及其局限性。前馈神经网络（FNN）由多个全连接神经元层堆叠而成。由于存在固定输入输出大小和梯度消失等问题，且可能无法充分利用时间特征，因此它并非最佳选择。卷积神经网络（CNN）是一类特殊的神经网络（NN），通过卷积层利用图像和时间序列中常见的维度特征。CNN 能够学习因果卷积（即感受野），这对准确预测至关重要。TCN 是专为时间预测设计的 CNN。循环神经网络（RNN）是一类专为处理序列数据设计的特殊神经网络。与 CNN 和 TCN 相比，RNN 更容易出现梯度消失或梯度爆炸。

Transformer 是一种基于注意力机制的现代架构，能够高效地执行长序列预测任务。与时间卷积网络相比，Transformer 仅在训练更长的预测序列时表现出更高的效率，而这并非既定目标所必需。图神经网络（图神经网络）可以通过学习时间嵌入以及其他嵌入来预测未来值。然而，只有在存在利用其图信息的策略时，它才优于时间卷积网络，而既定目标并不需要这一点。广义对抗网络（生成对抗网络）被用于预测，例如合成时间数据和衡量对抗时间损失。生成对抗网络通常与其他类别的模型结合使用，可能会使论文的模型设计和实现效率过于复杂。

考虑到上述 Benidis 等人 [14] 对用于时间预测的主要深度学习类别、实现效率以及各算法应对既定模型要求（MRs）能力的分析，本文选择时间卷积网络（TCN）进行实现。

A. TCN 实现

时间卷积网络使用函数所有者的 HTTP 触发函数实例到达历史数据、每个函数实例的平均执行时间，以及同一函数已配置的实例数量，来预测将配置哪些函数实例及其到达时间。预测范围

设为 500 次到达，相当于预测 Azure FaaS 未来 10 至 15 分钟的运行情况。这段时间足以处理任何后续冷启动管理任务。

在第一次交叉验证步骤（步骤 1）中，候选模型只会观察并学习前 500 个到达样本。在第二个验证步骤中，包含 500 个到达样本的窗口会向后滑动，直到第一步的末尾。模型将利用过去的知识（步骤 1 中 500 对 X 和 Y 值）以及当前步骤的 500 个 X 值，预测当前步骤的 500 个 Y 值（实例到达量）。进入步骤 3 之前，它会将预测的 Y 值与训练数据集中该步骤窗口对应的真实值 Y 进行验证。此过程会重复进行，直到到达训练数据集的末尾，或模型性能无法再提升为止。

本文的 TCN 实现包含三个主要组成部分。预混合层会将输入数据混合成一个信号数组。由多个膨胀卷积层组成的堆栈会处理混合后的输入信号，其机制如图 4 所示。每一层的输出都会生成一个新的通道数组，其中包含由膨胀卷积滤波后的信号混合而成的内容。这些输出将作为输入馈入上层堆栈。最后，预测层会汇合输出信号并生成预测结果。

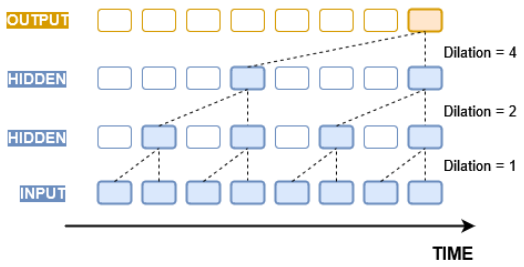


图 4. 膨胀机制

这些膨胀因果卷积层对于时间卷积网络的预测能力至关重要。卷积会混合并组合某个时间点周围的信号，而膨胀则决定了邻域的大小。当我们逐层堆叠时，顶层中的一个时间点将拥有更宽的卷积基底，覆盖更长的时间跨度。换句话说，模型的层数越多，它能“看到”的过去就越远。感受野 t_{rf} 可描述为 $t_{rf} = 4n_b(2_c^n - 1) + 1$，其中 n_b 为膨胀块的数量，n_c 为每个块中的单元数量。

B. 流水线

该流水线由两个串联的模块组成。模块A用于预测未来实例的函数名称，模块 B 用于预测这些实例的到达时间。这种设计提高了策略集成的灵活性，也支持持续运行和备用规划。具体而言，当模块A的主模型出现问题时，其备用模型可以无缝接替，生成模块 B 所需的实例函数名称。模块A和模块 B 的预测结果可以支持不同但互补的策略子集，相关内容将在后续章节中讨论。

每个模块包含一个或两个块。块的数量代表了 TCN 流水线的深度。每个块包含三种残差单元，其膨胀率以 2 为底呈指数递增。在每个残差单元中，我们会堆叠两组与图 4 类似的因果卷积层，随后进行归一化和非线性激活（ReLU）。每个块包含 3 到 6 个单元。隐藏通道的数量为 64 到 256。残差单元的最终输出由其堆叠层的输出与初始输入组合而成。图 5 直观地总结了该流水线。

采用自动机器学习策略来探寻最佳超参数集。对于每个模块，我们使用能够合理覆盖超参数空间的超参数集，构建了 12 个基准时间卷积网络模型，并对每个模型的性能进行了测量。随后，利用性能最佳的基准模型的超参数集，计算出用于定位最佳性能模型的精调超参数集。模型探索的时间预算设置为最多 80 小时，这是由于 Azure云个人使用的资源限制所致。最后，在同一数据集上，将性能最佳的模型与其他基于机器学习的预测模型进行了基准比较。图 6 展示了模块A（预测 HTTP触发实例的函数名称）和模块 B（预测 HTTP触发实例的到达时间）的自动超参数探索结果。

基于这些结果，未来希望采用这项工作的开发者和研究人员只需围绕所提供的性能最佳的 TCN模型超参数集进行探索即可。性能最佳的 TCN模型的具体性能结果将在下一节中进一步讨论。

V. 结果

针对两个模块的模型，均采用适用于时间序列预测任务的一整套指标进行了评估。模型在每个交叉验证步骤上进行评估，步骤数等于预测范围。为简洁起见，本文仅列出最后一个交叉验证步骤的评估结果。两个模块所有模型的完整评估结果可在 *https://github.com/Cybonto/TCN4CLOUD* 找到。主要指标如下。

解释方差表示原始数据方差相对于误差方差的百分比降幅。解释方差越接近 1，效果越好。平均绝对百分比误差（MAPE）衡量预测值与实际值之间的平均差异。MAPE 越接近 0，效果越好。均方根误差（RMSE）等于预测值与实际值之差的平方的均值再开平方。归一化均方根误差是 RMSE 除以数据范围。归一化均方根误差越接近 0，效果越好。R^2评分表示均方误差相对于数据总方差的降低程度。R^2越接近 1，效果越好。斯皮尔曼相关系数衡量两个数据集之间关系的单调性。斯皮尔曼相关系数越接近 1，效果越好。表II给出了

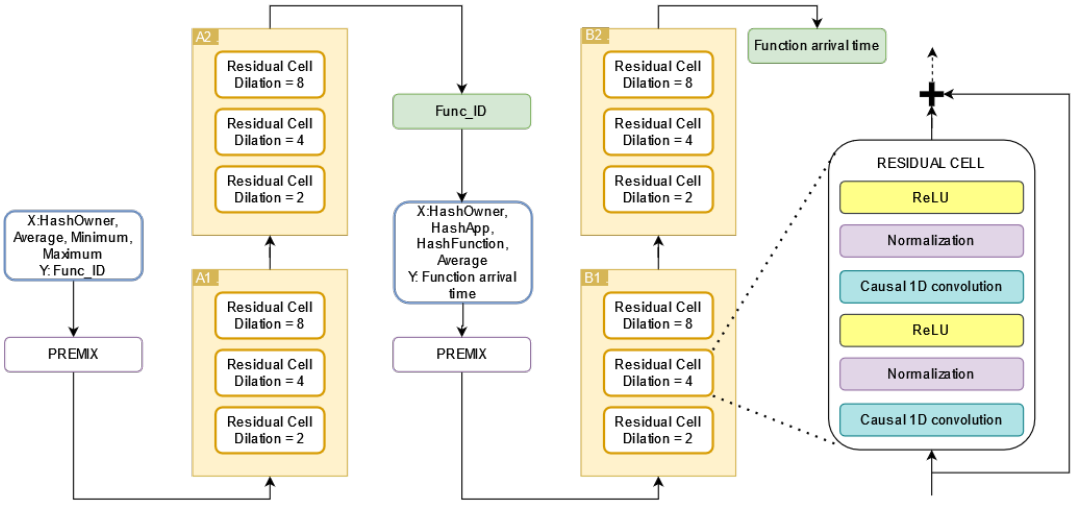
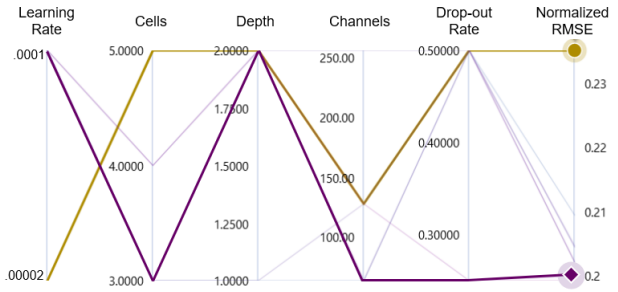
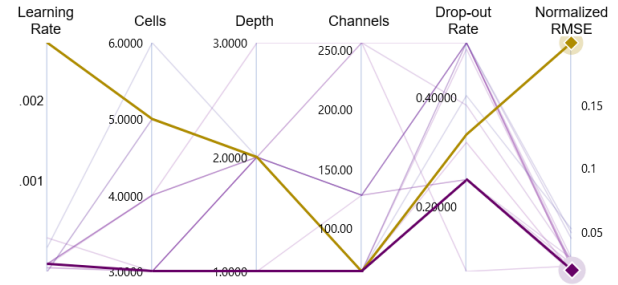


图5. 流水线设计



a. Hyper-parameter Exploration for Module A



b. Hyper-parameter Exploration for Module B

图6. 自动超参数探索结果

预测实例函数名称（模块A）的基准测试结果，表 III 给出了预测实例到达时间（模块 B）的基准测试结果。

A. 提出的集成策略

预测将调用哪个新函数实例（模块 A）比预测这些函数何时会被调用（模块 B）困难得多。主要原因是，创建新函数实例的需求取决于 FaaS环境状态和 FaaS用户的业务需求。后者的信息可能无法从过去的实例配置中获取。因此，构建一个

表II 预测实例函数名称的性能

指标	TCN	ARIMA	ES[b]	Prophet
解释方差	-.002	-.002	~0	.012
平均绝对百分比误差	1968	136	126	142
归一化均方根误差	.200	.238	.286	.220
R^2评分	-0.05	-0.29	-0.62	-0.12
斯皮尔曼相关系数	-.042	.078	-1	0.15

a 均方根误差：均方根误差。b 指数平滑：指数平滑。

表 III 实例到达时间预测性能

指标	TCN	Arimax	ARIMA	ES[b]	Prophet
解释方差	.50	.32	.20	~0	-.002
平均绝对百分比误差	15.79	230	210	291	252
归一化均方根误差[a]	.16	.18	.20	.20	.27
R^2评分	-.87	-.39	-.59	-.80	-.80
斯皮尔曼相关系数	.92	.32	.32	-1	-.05

a 均方根误差：均方根误差b 指数平滑：指数平滑

用于模块 A 的时间卷积网络模型可能代价高昂，尤其是在已注册函数数量较多时。在本研究中，当模型深度为 2 且隐藏通道超过 150 个时，Microsoft Azure 机器学习的默认计算配额不足以避免模块 A 的时间卷积网络模型训练崩溃。因此，表II中展示的是表现最佳的基准模型的结果，而非模块 A 的微调模型。即便如此，该基准模型的表现仍可与最佳的非深度学习模型（Prophet）相媲美。资源更多的公司或许可以使用相同的代码，但采用不同的超参数，构建出性能好得多的时间卷积网络模型。

如表 III 所示，在已知未来实例的函数名称时，时间卷积网络在预测实例到达时间方面优于非深度学习模型。由于最佳模型的

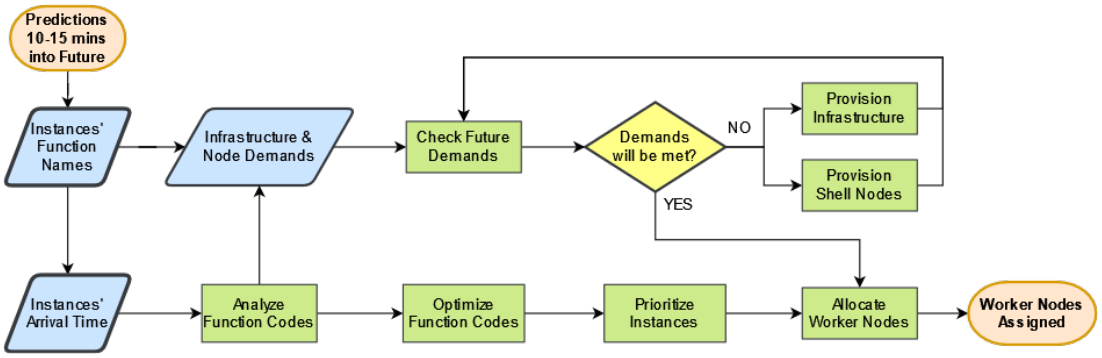


图 7. 集成冷启动管理策略

超参数仅要求深度为 1 且隐藏通道数少于 80，因此可以在 Microsoft Azure 机器学习上针对模块 B 任务训练时间卷积网络。

VI. 提出的策略

由于云无服务器冷启动问题的根本原因涉及无服务器技术栈的不同层面，因此解决这一问题并非易事。因此，本文提出了一种集成策略，采用分而治之和并行处理策略。基于先前研究，本文将冷启动子问题分为两组：基础设施层问题和函数层问题。假设 TCN 模型能够如前文所述，可靠地预测未来 10 至 15 分钟的情况，那么我们就有足够的时间集成各种冷启动管理策略，并对其进行编排和自动化。预测的函数名称可用于解决基础设施层问题，而预测的函数到达时间可用于解决函数层问题。得益于 TCN 流水线松耦合、高内聚的设计，本文提出的策略能够同时解决这两个层面上的部分问题。具体细节如下。

A. 先前研究

先前研究表明，工作节点及相关支撑基础设施的设置时间成本高昂 [8], [15]。解决方案包括预先配置网络缓存和 Shell节点 [15], 预先配置新的 Shell节点，并在物理集群之间对现有工作节点进行负载均衡[8], 对服务器进行排名和选择，以进行函数调度 [16], 以及根据节点缓存未命中率预先配置工作节点 [17]。保活策略可以基于成本、频率、时间、大小等运行优先级 [17]。

在函数级，客户视角（即客户注册函数时不会指定函数优先级）和云服务提供商视角（即在满足执行约束的同时，函数调度并未达到成本最优）下，函数执行优先级都不明确[8]。相关解决方案包括分析依赖关系、截止时间 [8],、环境状态、动作、奖励及其他指标，以预先按恰当的顺序配置和终止函数支持资源 [6], [12], [18]。具体而言，模型可以根据依赖关系可预测函数的调用直方图来设定函数的预热时间和保活时间，同时对依赖关系不可预测的函数采用固定超时策略 [19]。还可以通过优先加载关键函数、稍后加载可选函数来缩短应用程序代码加载延迟 [20],，或者通过融合并优化函数代码来提高运行时执行效率，从而解决函数级冷启动问题 [21], [22]。

B. 集成策略

由于冷启动问题根源于无服务器云的多个层面，因此最好同时从多个角度着手解决。基于 TCN 能够预测未来 10 至 15 分钟内函数实例到达情况的能力，本文提出了一种利用前文所述相关研究成果的集成策略。具体而言，该策略包含两条并行路径（A 和 B），描述如下。

在路径 A 中，从模块 A 模型接收到预测实例的函数名称后，即可推断未来（接下来 10 至 15 分钟）的基础设施和工作节点需求。这是因为所有函数都应已预先注册，从而可以将函数名称与系统需求进行粗粒度映射。更细粒度的详细信息则可由路径 B 对函数代码进行深入分析后提供。如果未来需求无法得到满足，则会根据 [6], [8], [12], [15], [18]预先配置更多基础设施和/或 Shell节点。如果未来需求可以得到满足，系统将根据 [16]对现有可用资源进行排名并进行编排。资源排名和基础设施编排调度计划将与路径 B 共享。

在路径 B 中，将根据预测的到达时间分析预测函数实例的代码。假设客户向 CSP 注册函数时已完成代码分析，则此过程可以达到磁盘 I/O 读取速度。由此可以推断出更细粒度的代码依赖关系，并将其传递给路径 A，以辅助未来的需求推断。随后，可以根据 [21], [22] 优化函数代码。基于函数实例的到达时间、细粒度需求和优化后的函数代码图，可以对实例进行优先级排序。例如，预测最先到达的实例不一定会最先获得所需资源。在最后一步，路径 B

将其计算出的所有信息与路径 A 的资源排名和基础设施编排调度计划相结合，以决定哪些 Shell 节点将转换为哪些工作节点。图 7 总结了该策略。

VII. 结论

无服务器云计算是一种颇具吸引力的云服务模型，因为它能以其他云服务模型的一小部分成本，为企业提供同样的云优势。无服务器云面临的最大问题之一，是如何在关闭闲置云资源以降低成本，与保留闲置资源以便及时执行客户请求之间取得平衡。如果没有现成的云资源来处理客户请求，就必须配置新的资源。当资源配置过程耗时过长、无法接受时，便会出现冷启动问题。最先进的解决方案从无服务器云不同层面入手，以不同角度应对冷启动问题。现有研究的不足主要在于对无服务器云工作负载作出了不切实际的假设或模拟，以及关注范围过于狭窄。

本文以更加全面的方式应对冷启动问题。首先，识别了无服务器云环境中的一系列实际挑战，并将其转化为数据集要求和预测模型要求。基于这些要求，我们实现了一个时间卷积网络模型，可预测未来 10 至 15 分钟内的函数实例及其到达情况。在此基础上，我们构建了一种集成策略。该策略兼顾并行性，遵循低耦合高内聚原则，并利用相关先前研究成果，在基础设施层和函数层应对冷启动问题。除了冷启动管理之外，所提出的策略和公开可用代码还可用于解决其他云问题，例如优化虚拟软件定义网络资产的配置过程。

参考文献

[1] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, “真实环境中的无服务器：表征并优化大型云服务提供商的无服务器工作负载”，*2020* 年 *USENIX* 年度技术会议论文集，*ATC 2020*，第 205–218 页，2020 年。[2] H. Shafiei, A. Khonsari, and P. Mousavi, “无服务器计算：机遇、挑战与应用综述”，*ACM* 计算调查，第 54 卷，第 11s 期，第 1–32 页，2022 年。[3] A. Tayal, E. Lam, D. Choudhury, M. Dickerson, G. Moovera, and G. Arora, “与传统云相比，无服务器技术的总拥有成本”，德勤咨询技术报告，2019 年 9 月。[4] N. Mahmoudi and H. Khazaei, “无服务器计算平台的性能建模”，*IEEE* 云计算汇刊，第 10 卷，第 4 期，第 2834–2847 页，2020 年。[5] H. Ebrahimpour, “一种基于启发式的包感知函数调度方法，在 FaaS 计算环境中实现冷启动时间与成本之间的权衡”，2022 年。[6] A. P. Jegannathan, R. Saha, and S. K. Addya, “一种在云无服务器平台中最小化冷启动时间的时间序列预测方法”，*2022* 年 *IEEE* 国际黑海通信与网络会议，*BlackSeaCom 2022*，第 325–330 页，2022 年。

[7] M. Steinbach, A. Jindal, M. Chadha, M. Gerndt, and S. Benedict, “TppFaaS：通过时间点过程对无服务器函数调用进行建模”，*IEEE Access*，第 10 卷，第 9059–9084 页，2022 年。[8] A. Singhvi, A. Balasubramanian, K. Houck, M. D. Shaikh, S. Venkatara- man, and A. Akella, “Atoll：一种可扩展的低延迟无服务器平台”，*SoCC 2021 - 2021* 年 *ACM* 云计算研讨会论文集，第 138–152 页，2021 年。[9] C. Lin and H. Khazaei, “无服务器应用的性能与成本建模及优化”，*IEEE* 并行与分布式系统汇刊，第 32 卷，第 3 期，第 615–632 页，2021 年。[10] A. Das, S. Imai, S. Patterson, and M. P. Wittie, “通过动态任务放置优化边缘云无服务器平台的性能”，第 *20* 届 *IEEE/ACM* 集群、云和互联网计算国际研讨会论文集，*CCGRID 2020*，第 1 期，第 41–50 页，2020 年。[11] A. Mampage, S. Karunasekera, and R. Buyya, “无服务器计算环境中的资源管理全景：分类体系与未来方向”，*ACM* 计算调查，第 54 卷，第 11s 期，第 1–36 页，2022 年。[12] S. Agarwal, M. A. Rodriguez, and R. Buyya, “一种通过强化学习降低无服务器函数冷启动频率的方法”，第 *21* 届 *IEEE/ACM* 集群、云和互联网计算国际研讨会论文集，*CCGrid 2021*，第 797–803 页，2021 年。[13] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, “Hermod：兼具原则性与实用性的无服务器函数调度”，*SoCC 2022* - 第 *13* 届云计算研讨会论文集，第 289–305 页，2022 年。[14] K. Benidis, S. S. Rangapuram, V. Flunkert, Y. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, F. X. Aubet, L. Callot, and T. Januschowski, “时间序列预测的深度学习：教程与文献综述”，*ACM* 计算调查，第 55 卷，第 6 期，2022 年。[15] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhom- linov, “面向可扩展无服务器系统的敏捷冷启动”，第 *11* 届 *USENIX* 云计算热点主题研讨会，*HotCloud 2019*，与 *USENIX ATC 2019* 联合举办，2019 年。[16] H. Yu, A. A. Irissappane, H. Wang, and W. J. Lloyd, “FaaSRank：学习如何调度无服务器平台中的函数”，*2021* 年 *IEEE* 自治计算与自组织系统国际会议论文集，*ACSOS 2021*，第 31–40 页，2021 年。[17] A. Fuerst and P. Sharma, “FaasCache：通过贪婪双重缓存保持无服务器计算的运行”，编程语言与操作系统架构支持国际会议，*ASPLOS*，第 386–400 页，2021 年。[18] D. Bermbach, A. S. Karakaya, and S. Buchholz, “利用应用知识减少 FaaS 服务中的冷启动”，*ACM* 应用计算研讨会论文集，第 134–143 页，2020 年。[19] J. Shen, T. Yang, Y. Su, Y. Zhou, and M. R. Lyu, “Defuse：一种依赖关系引导的函数调度器，用于缓解 FaaS 平台上的冷启动”，分布式计算系统国际会议论文集，第 2021 年 7 月卷，第 194–204 页，2021 年。[20] J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, X. Jin, and X. Liu, “LambdaLite：通过应用级优化降低无服务器计算中的冷启动延迟”，2022 年。 [在线]。可访问：http://arxiv.org/abs/2207 08175 。T. Schirmer, J. Scheuner, T. [21] Pfandzelter, and D. Bermbach, “Fusion- ize：通过反馈驱动的函数融合提升无服务器应用性能”，*2022* 年 *IEEE* 云工程国际会议论文集，*IC2E 2022*，第 85–95 页，2022 年。[22] S. Lee, D. Yoon, S. Yeo, and S. Oh, “在无服务器计算中通过函数融合缓解冷启动问题”，*Sensors*，第 21 卷，第 24 期，2021 年。