Serverless 计算模式下的冷启动问题研究：现状、挑战与展望

梁俊轩[1)]

[1)](计算机科学与技术，电子信息与电气工程学院, 上海交通大学, 上海市, 中国)

摘　要　本文聚焦于云计算中 Serverless 计算模式下的冷启动问题，通过对近十年相关文献的广泛调研，深入剖析其主要特点与挑战。Serverless 函数粒度小、执行时间短，应用程序规模多样，内存需求差异大，触发和调用模式复杂多变，且具有时变性和突发性，这些特点使得冷启动问题的解决极具挑战性。目前前沿解决方案包括实例复用、预热和资源池化，但各有缺陷。未来研究方向涵盖编程语言优化、虚拟化改进、并发与实例处理优化等方面，旨在实现云计算节能与用户体验的平衡，为相关领域研究和实践提供全面参考。

关键词　Serverless 计算，冷启动问题，云计算技术

„

1　引言

在当今数字化时代, 云计算技术持续演进, Serverless 计算模式作为其中的新兴力量，正逐渐崭露头角。如图 1 所示，Serverless 计算，亦被称为服务器无感知计算或函数计算。其核心使命在于极大程度地简化云业务的开发流程，使得应用开发者得以从繁琐复杂的服务器运维事务中脱身，例如监控、自动伸缩、日志管理等工作。借助 Serverless 计算模式，开发者仅需上传业务代码，并进行简易的资源配置，即可快速实现服务的构建与部署[1]。在此过程中，云服务商依据函数服务调用量以及实际资源使用情况收取费用，有力地协助用户达成业务的快速交付，同时实现低成本运行的目标。

在当前的技术发展阶段，业界广泛认可的 Serverless 计算的精确界定为“FaaS+BaaS”，即 Function-as-a-Service（函数即服务）与 Backend-as-a-service（后端即服务）的有机组合。这种计算模式深刻地变革了用户运用云资源的方式。为实现细粒度的资源精准供给以及高度弹性化的扩缩容能力，Serverless 计算引入了无状态函数的编程抽象概念。具体而言，就是将用户的应用程序拆解并构建为多个相互独立、无状态的函数集合，随后通过中间存储、BaaS 等各类业务组件进行交互协作，进而构建出完整的云应用[2]。

在无服务器计算的运行机制中，一个服务请求存在两种截然不同的处理途径。其一，使用一个全新的容器来处理请求，此过程涉及一系列复杂的准备工作，包括启动容器、初始化特定的运行时环境（例如 Python 语言的运行时）以及加载相关的功能代码等，这一过程被定义为冷启动，如图 2 所示。其二，若使用一个已经预先准备就绪的容器，则可直接进行服务处理，此为热启动。值得注意的是，大部分函数的执行时间相对短暂，其时长与冷启动时间处于同一数量级[3]，通常在几十毫秒到数秒之间。然而，在核心业务链路层面，情况更为复杂。用户不仅需要运行自身的业务逻辑，还高度依赖于中间件、数据库、存储等后端服务。这些后端服务的连接建立操作必须在实例启动阶段同步完成，这无疑在无形中大幅增加了冷启动所需的时间，甚至可能将冷启动时间延长至秒级水平。相反，若始终保持容器处于热启动状态，虽然能够确保快速响应用户请求，但却会引发大量的资源闲置与浪费[4]。对于无服务器计算服务提供商而言，他们期望在用户眼中营造出资源始终处于热启动状态的假象，以此保障服务质量，同时又渴望资源在实际运行中尽量保持冷启动状态，以降低资源消耗。因此，如何在尽可能降低成本的前提下，有效减少冷启动发生率，实现高水准的服务质量，已然成为无服务器计算提供商关注的焦点问题。

本文对近十年来关于 serverless 计算中的冷启动问题文献进行了广泛的调研，文中对不同 serverless 框架对于冷启动问题的优化进行了详细论述分析，并指出未来研究方向，进而实现云计算节能与用户体验的平衡。

本文的贡献有：(1) 分析 serverless 计算中的冷启动问题。(2) 分析各类解决方案遇到的挑战。(3) 展望 serverless 计算中的冷启动问题的未来。

2　冷启动问题的主要特点与挑战

- 执行时间与函数粒度：与传统虚拟机（VM）中的应用相比，Serverless 函数粒度更小，执行时间大

E-mail: whitefork@sjtu.edu.cn.

幅缩短，甚至与冷启动时间处于同一数量级。

- 应用程序规模：绝大多数 Serverless 应用程序规模较小，这种小规模的应用结构使得开发和部署更加灵活便捷，但也对资源管理和调度提出了更高的要求。
- 函数内存资源需求差异：不同的 Serverless 函数在内存资源需求方面存在巨大差异。
- 触发方式的多样性：Serverless 函数存在七种触发方式, 包括 HTTP（通过点击外部链接触发）、Event、Queue、Timer（定时器触发，可同时存在多个定时器）、Orchestration（函数工作流的构成方式）、Storage（数据变更触发）以及 others（其他）。
- 调用模式的多样化：
  - 调用频率差异巨大, 跨越 8 个数量级, 这使得资源分配需根据不同的调用频率进行调整。
  - 多种触发方式进一步丰富了调用模式。不同的触发方式会导致不同的调用模式，例如 HTTP 触发可能是随机的用户请求, 而 Timer 触发可能是周期性的。
  - 应用场景的多样化也是调用模式多样化的重要因素。比如，IOT 应用可能具有周期性触发的情况[5]，而 Web 应用则可能具有随机的用户请求触发。由于这些因素的综合影响，调用模式很难用泊松分布等传统概率分布来准确刻画。
- 调用模式的时变性：调用模式会随时间发生迁移和改变。这种时变性要求资源管理系统能够动态适应变化，及时调整资源分配策略。
- 调用请求的突发性：指在一段时间内调用请求急剧增长的情况。这就需要弹性缩放机制能够在短时间内快速增加资源以满足突发需求，然后在需求下降时及时回收资源。

3 解决方案

3.1 实例复用

目前，商用的 Serverless 平台，如 AWS Lambda[6][7]、Azure[8]、华为云 FunctionGraph[9]、Google Cloud Functions[10]，都内置了 Time-to-live 实例保活机制，也就是 keepalive 方法。其原理是在函数执行完成后，将执行完请求的实例在内存中保活一段时间再进行释放。这一机制旨在平衡冷启动问题与资源消耗问题。通过合理设置保活时间，既能在一定程度上减少冷启动的发生，提高系统的响应速度，又能避免过长时间的资源占用，降低资源消耗成本。

在实例资源复用方案中，确定实例要被保留多长时间是一个关键问题。如果保留时间过短，后续服务请求中可能会频繁出现较为严重的冷启动情况，这将极大地影响用户体验，尤其对于那些对响应时间要求较高的应用场景，如实时交互应用、金融交易系统等。用户可能因冷启动导致的响应延迟而不满，甚至影响业务正常进行。另一方面，如果保留时间过长，虽然可以减少冷启动的发生概率，但会造成极高的资源消耗，在函数调用率较低时尤为明显。大量实例被长时间保留，占用宝贵计算资源却可能长时间处于闲置状态，无疑是一种资源浪费，也会增加运营成本。

3.2 实例预热

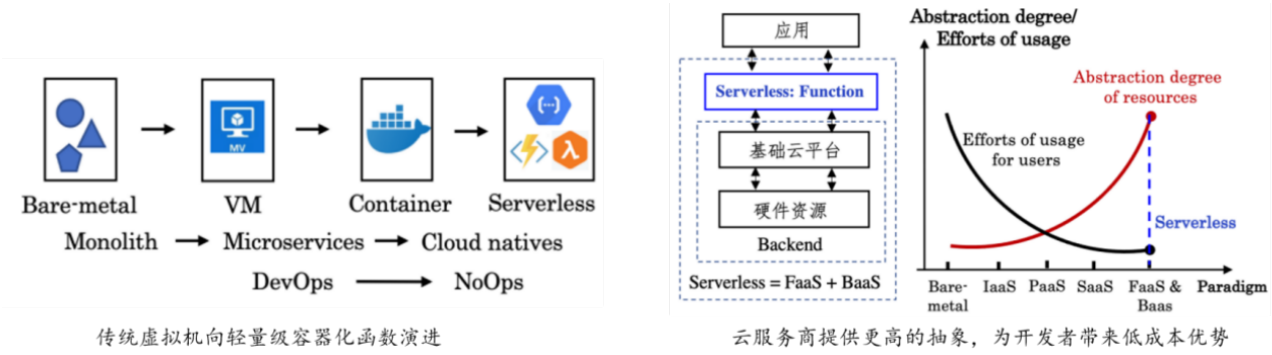
在 AWS 中，用户可以设置两种预热方式[6]。其一为借助 Amazon EventBridge 按照等时间间隔生成事件。这种方式较为适合事件较少且同一时间段内事件请求不会太多的应用场景。然而，当同一时间段调用 Lambda 的事件过多时，预热的函数可能无法满足响应需求，此时 AWS 就会新增 Lambda 实例，这同样会导致冷启动问题，进而增加响应时间。其二是设置预制并发，该功能能够明确处于激活状态的 Lambda 实例数量，让函数始终保持初始化状态。CloudWatch[11]、Thundra[12]、Dashbird[13] 和 lambda Warmer[14] 通过定期发送服务请求减少冷启动的发生。OpenFaaS[15] 为开发者提供了一个选项，即为一个函数使用一个始终运行的容器的功能。FAST-BUILD[16] 利用文件级镜像缓存复用机制和并行化镜像构建技术，加快容器构建速度。阿里云的 FaasNet[17] 基于树结构快速分发跨节点镜像。YiFan Sui[18] 提出了一种名为 InstaInfer 的技术, 可以在容器预热的同时, 预先加载所有必要的机器学习库和模型，从而实现几乎无延迟的启动。

实例预热也会存在两个问题：第一，实例数量较多的和较少的情况下会出现阈缩数量过大（资源浪费）或过小（冷启动发生率高）的问题，所以有部分厂商引入不同实例范围内采用不同的比例来解决这个问题[6][12][11]；第二，在流量波动较频繁且波峰波谷相差较大的时候会出现预热滞后的问题（应对负载突发的能力差）；

3.3 资源池化

资源池化在于实例所需的相关资源并非完全准备就绪，而是在实例生成过程中的任意步骤都进行了一定程度的预留准备。这种池化策略既可以精细到用户代码层面，也可以将底层资源、运行时资源等作为池化资源进行准备。

Fission[19] 和 Knative[20] 平台采用的是平台的现



传统虚拟机向轻量级容器化函数演进　　云服务商提供更高的抽象，为开发者带来低成本优势

图 1　Serverless 架构

图 2　冷启动流程

成容器池：这些平台有一个始终运行的容器池，可以减少延迟并提高效率。但是这种方法不具有资源成本效益。Openwhisk[21] 平台使用预热的干细胞容器，并根据内存和编程语言对容器进行分类。RemoteFork[22] 采用基于 RDMA 共享内存减少跨节点启动过程的数据拷贝，华为云 FunctionGraph[9] 采用 Pod 池 + 特化实例跳过镜像传输，AWS[6] 使用基于 Block 和分布式多级缓存的镜像分发策略。其中，快速镜像分发依赖于 VM 节点的上/下行网络带宽，Pod 池特化技术则是典型的以空间换时间的做法。

然而，资源池化也面临着一些挑战。池化资源的规格、运行时的种类，以及池化的数量、资源的分配和调度等问题，都是各个厂商在对自身冷启动问题进行优化时必须考虑的重要因素。

3.4　其他解决方案

SAND（应用沙盒）[23] 通过解除同一应用程序内不同函数之间的隔离性，同时保持不同应用程序之间的隔离，减少了资源消耗和时间开销，大大降低了冷启动时间，但是由于分配的资源有限而导致函数执行出现延迟。Shahrad[24] 考虑使用一个依赖于具体函数预热窗口参数和保活窗口参数分别对预热和重用过程进行控制，但是这种基于参数预测的方式难以应对负载突发的状况。Vahidinia[25] 提出了一种两层自适应方法来解决冷启动，同时采用了预热和重用策略。第一层利用强化学习算法随着时间的推移发现函数调用模式，以确定保持 keep-alive 的最佳时间。第二层是基于长短期记忆（LSTM），用于预测未来的函数调用时间，以确定所需的预热容器数目。但是这种方式其实缺少对于容器生命周期的感知，同时这种批量预热的方式并不利于资源的高效利用。

3.5　解决方案之间的对比

如表1所示，现在主流的不同框架之下解决方案的采用也不同，主流的框架注重于采用资源池化、实例预热、实例复用这些方法来解决冷启动问题，这些解决方案占用的缓存资源较多。各种方案的优缺点对比如表2所示，可以看到目前各种解决方案都存在一定的优点和缺陷，如何在运营成本和时间成本之间达到一个平衡，也是亟待研究的。

4　未来的研究方向

4.1　编程语言层面的优化

针对特定的编程语言，开发专门的冷启动优化工具和库。推广使用编译型语言或具有提前编译功能的语言，对于一些对启动速度要求较高的场景，使用编译型语言可以避免解释型语言在运行时的解释执行过程，从而大大提高函数的启动速度。例如，使用 Go 语言、Rust 语言等编译型语言来编写函数，或者利用 Java 的提前编译技术（如 GraalVM）将函数代码提前编译成本地代码，以减少冷启动时间。

4.2　虚拟化

虚拟化技术能够以极低的开销和极快的启动、关闭速度创建虚拟环境。在函数需要启动时，可迅速基于这些虚拟环境进行实例化，跳过复杂的底层系统初始化过程，极大地缩短了冷启动过程中创建运行环境的时间。

4.3　改善并发/实例共享或复用

可以采用一种支持多并发处理的函数实例运行方案，通过为实例分配多个独立的 CPU 处理线程来提高实例的吞吐。同时，通过计算实例的排队延迟，并发处理数和未来请求到达率来提前预置对应数量的实例，从而减少冷启动调用的产生。

平台/方案	减少实例准备延迟	调用函数链	资源池化	实例预热	实例复用	适应性
Openwhisk[21]	✓	✗	✓	✗	✓	✗
SAND[23]	✓	✓	✓	✗	✗	✗
CloudWatch[11]	✗	✗	✗	✓	✗	✗
Thundra[12]	✗	✗	✗	✓	✗	✗
Dashbird[13]	✗	✗	✗	✓	✗	✗
lambda Warmer[14]	✗	✗	✗	✓	✗	✗
OpenFaaS[15]	✗	✗	✗	✓	✓	✗
Knative[20]	✗	✗	✓	✗	✓	✗
Fission[19]	✗	✗	✓	✗	✓	✗
AWS Lambda[6][7]	✗	✗	✓	✗	✓	✗
RemoteFork[22]	✗	✗	✓	✗	✓	✗
FaasNet[17]	✗	✗	✗	✓	✓	✗
FAST-BUILD[16]	✗	✗	✗	✓	✓	✗
Azure[8]	✗	✗	✗	✗	✓	✗
Shahrad[24]	✗	✗	✗	✓	✓	✗
Vahidinia[25]	✓	✗	✗	✗	✓	✓
华为云 FunctionGraph[9]	✗	✗	✓	✗	✓	✗
InstaInfer[18]	✗	✗	✗	✓	✗	✗

表 1　不同框架对比表

方案	优点	缺点	特性
减少实例准备延迟	直接提升了函数的响应速度	涉及系统重构，成本较高	快速为实例分配所需资源
调用函数链	有助于实现函数的复用	增加了系统的耦合性	定义函数的依赖关系
资源池化	降低冷启动的概率	资源池的管理需要一定的开销	动态资源调整
实例预热	有效地减少冷启动的发生	难以精确预测负载情况	根据负载变化调整预热策略
实例复用	显著降低冷启动的频率	导致数据安全和隔离性问题	实例管理

表 2　不同方案对比表

此外，还可以采用一种支持多并发处理的函数实例运行方案。通过为实例分配多个独立的 CPU 处理线程，能够显著提高实例的吞吐量。同时，通过计算实例的排队延迟、并发处理数以及未来请求到达率，可以提前预置相应数量的实例，从而减少冷启动调用的产生。这种方法能够根据实际的负载情况动态地调整实例的数量，既能够保证系统的性能，又可以避免资源的浪费。

5　总结

Serverless 的无状态设计赋予了函数计算高度弹性化的扩展能力，然而也带来了难以避免的冷启动问题。解决冷启动问题的解决方案有非常多种，但都不可避免地存在一部分缺陷，且现在主流的解决方案便是采用空间换时间的方式，增加缓存，减少冷启动时间，这不可避免地提高了企业的运营成本。未来的解决方案应该综合考虑各种现有的研究方案，为客户和企业之间实现一个双赢的效果。

致　谢　感谢李超老师和侯小风老师为我解惑，让我对云计算技术有了更深入的理解。另外也感谢上海交通大学电子信息与电气工程学院自动化系 2021 级博士研究生隋奕帆，在关注到他关于无服务计算的论文“Pre-Warming is Not Enough: Accelerating Serverless Inference With Opportunistic Pre-Loading”获评 SoCC 2024 会议最佳论文奖（Best Paper Award）后，便于其积极沟通，对本文写作颇多帮助。

参考文献

[1] SEGARRA C, FELDMAN-FITZTHUM T, BUONO D, et al. Serverless confidential containers: Challenges and opportunities [C/OL]//SESAME '24: Proceedings of the 2nd Workshop on

SErverless Systems, Applications and MEthodologies. New York, NY, USA: Association for Computing Machinery, 2024: 32–40. https://doi.org/10.1145/3642977.3652097.

[2] KOUNEV S, HERBST N, ABAD C L, et al. Serverless computing: What it is, and what it is not?[J/OL]. Commun. ACM, 2023, 66(9): 80–92. https://doi.org/10.1145/3587249.

[3] SILVA P, FIREMAN D, PEREIRA T E. Prebaking functions to warm the serverless cold start[C/OL]//Middleware '20: Proceedings of the 21st International Middleware Conference. New York, NY, USA: Association for Computing Machinery, 2020: 1–13. https://doi.org/10.1145/3423211.3425682.

[4] GOLEC M, WALIA G K, KUMAR M, et al. Cold start latency in serverless computing: A systematic review, taxonomy, and future directions[J/OL]. ACM Comput. Surv., 2024. https://doi.org/10.1145/3700875.

[5] PINTO D, ANDRé R, PEDRO R. Serverless architectural design for iot systems[D]. [S.l.]: Universidade do Porto (Portugal), 2018.

[6] PELLE I, CZENTYE J, DóKA J, et al. Dynamic latency control of serverless applications operated on aws lambda and greengrass[C/OL]//SIGCOMM '20: Proceedings of the SIGCOMM '20 Poster and Demo Sessions. New York, NY, USA: Association for Computing Machinery, 2021: 33–34. https://doi.org/10.1145/3405837.3411381.

[7] VILLAMIZAR M, GARCéS O, OCHOA L, et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures[C/OL]//CCGRID '16: Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. IEEE Press, 2016: 179–182. https://doi.org/10.1109/CCGrid.2016.37.

[8] DHALLA H K. Performance testing of a web application using azure serverless functions and apache jmeter[J]. J. Comput. Sci. Coll., 2023, 39(3):26.

[9] BLUESKY. FunctionGraph Serverless 计算服务[EB/OL]. 2024. https://zhuanlan.zhihu.com/p/468964589.

[10] DONGHUI. Google 的 Serverless 产品对比[EB/OL]. 2024. https://zhuanlan.zhihu.com/p/349990127.

[11] TEAM D. Amazon cloudwatch events user guide[M]. London, GBR: Samurai Media Limited, 2018.

[12] VIANNA A, KAMEI F K, GAMA K, et al. A grey literature review on data stream processing applications testing[J/OL]. J. Syst. Softw., 2023, 203(C). https://doi.org/10.1016/j.jss.2023.111744.

[13] LIDDLE P. dashbird[EB/OL]. 2024. https://dashbird.io/.

[14] ALTSCHULER J M. lambda-warmer[EB/OL]. 2024. https://github.com/jeremydaly/lambda-warmer.git.

[15] ELLIS A. openfaas[EB/OL]. 2024. https://www.openfaas.com/blog/introducing-openfaas-for-lambda/.

[16] FULIN F. fastbuild[EB/OL]. 2024. https://fastbuild.org.

[17] WANG A, CHANG S, TIAN H, et al. FaaSNet: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute[C/OL]//2021 USENIX Annual Technical Conference (USENIX ATC 21). USENIX Association, 2021: 443-457. https://www.usenix.org/conference/atc21/presentation/wang-ao.

[18] SUI Y, YU H, HU Y, et al. Pre-warming is not enough: Accelerating serverless inference with opportunistic pre-loading[C/OL]//SoCC '24: Proceedings of the 2024 ACM Symposium on Cloud Computing. New York, NY, USA: Association for Computing Machinery, 2024: 178–195. https://doi.org/10.1145/3698038.3698509.

[19] DGOUJARD. fission[EB/OL]. 2024. https://fission.io/.

[20] Knative[EB/OL]. 2024. https://knative.dev/docs/.

[21] openwhisk[EB/OL]. 2024. https://openwhisk.apache.org/.

[22] WEI X, LU F, WANG T, et al. No provisioned concurrency: Fast RDMA-codesigned remote fork for serverless computing[C/OL]//17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23). Boston, MA: USENIX Association, 2023: 497-517. https://www.usenix.org/conference/osdi23/presentation/wei-rdma.

[23] GIAS A U, CASALE G. Cocoa: Cold start aware capacity planning for function-as-a-service platforms[C/OL]//2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). 2020: 1-8. DOI: 10.1109/MASCOTS50786.2020.9285966.

[24] SHAHRAD M, FONSECA R, GOIRI I, et al. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider[C/OL]//2020 USENIX Annual Technical Conference (USENIX ATC 20). USENIX Association, 2020: 205-218. https://www.usenix.org/conference/atc20/presentation/shahrad.

[25] VAHIDINIA P, FARAHANI B, ALIEE F S. Mitigating cold start problem in serverless computing: A reinforcement learning approach[J/OL]. IEEE Internet of Things Journal, 2023, 10(5):3917-3927. DOI: 10.1109/JIOT.2022.3165127.