无服务器云计算综述

王灏洲 [1)]

[1)](华中科技大学计算机科学与技术学院, 武汉　中国　430074)

摘　要　近年来，一种新型云计算模式——无服务器计算受到了业界和学术界的广泛关注，正在迅速地发展。无服务器计算依托容器技术，提供了高并发、高兼容的特性，为开发者隐藏了底层服务器的细节，同时采取了更经济的按调用次数或时间计费的服务运营模式。无服务器计算拥有自动伸缩性和精细的计费粒度的优点，对于不断增长的网络服务需求具有更好的适应性。本文首先介绍无服务器计算的概念，然后阐述无服务器计算的优势与挑战，最后引入几种先进的无服务器计算领域的研究工作。

关键词　无服务器；云计算；集群；容器

A Survey of Serverless Cloud Computing

Haozhou Wang[1)]

[1)](Department of Computer Science and Technology, HuaZhong University of Science and Technology,Wuhan, China)

Abstract　In recent years, a new cloud computing model - serverless computing has received extensive attention from the industry and academia, and is developing rapidly. Relying on container technology, serverless computing provides high concurrency and high compatibility, hides the details of the underlying server for developers, and adopts a more economical service operation model that is billed by the number of calls or time. Serverless computing has the advantages of automatic scalability and fine billing granularity, and has better adaptability to the ever-growing demand for network services. This paper first introduces the concept of serverless computing, then explains the advantages and challenges of serverless computing, and finally introduces several advanced research work in the field of serverless computing.

Keywords　Serverless; Cloud Computing; Cluster; Container

1 引言

传统的基础设施即服务 (Infrastructure as a service, IaaS) 部署模式需要长期运行的服务器来实现可持续的服务交付，无论用户应用程序是否正在运行，这种独占分配都需要保留资源。因此，它存在资源利用率低的缺点。为了获得更高的资源利用率和更低的云计算成本，无服务器计算逐渐发展起来。亚马逊、谷歌、微软、IBM、阿里巴巴等大多数大型云厂商都已经提供了这种弹性计算服务。

2015 年，亚马逊推出了 AWS Lambda。Lambda 提供了云函数，并引起了对无服务器计算的广泛关注。云用户只需编写代码，而所有服务器配置和管理任务则留给云提供商。Faas（Function as a Service，函数即服务）是无服务器计算的核心，与此同时云平台还提供专门的无服务器框架，以满足特定应用程序的需求，如 BaaS（Backend as a Service，后端即服务）产品。Faas 模型实现了函数隔离和调用，而 Baas 提供了对在线服务的整体后端支持。

在无服务器平台中，用户只需使用高级语言编写云函数，选择应该触发函数运行的事件，例如将图像加载到云存储或将图像缩略图添加到数据库中，其余的工作都可以让无服务器系统处理：实例选择、扩展、部署、日志记录等。

无服务器计算的盛行，背后存在三个原因。首先，无服务器计算开辟了一种新的构建和量化应用程序和服务的方法，开发者可以打破传统的基于单机服务器的应用程序的舒服，能够使用粒度更细的云函数，这使得开发者可以集中精力于核心应用的逻辑，而服务器资源管理等工作则由无服务器平台进行。其次，无服务器计算的计费是根据与执行相关的某个维度（例如执行时间），而不是根据基础云平台的维度（例如分配的 VM 的大小和数量）进行的，用户可以节省开支。最后，采用多个云函数开发应用具有模块化的优势。对于云服务提供者而言，使用无服务器计算也有助于对资源进行更高效的管理。

2 背景

2.1 无服务器计算的架构

无服务器云的架构如 1所示。无服务器层位于应用程序和基础云平台之间，简化了云编程。其中 FaaS 提供通用计算，并辅以专门的 BaaS 产品生态系统，例如对象存储、数据库或消息传递。无服务器计算还包括某些大数据服务，例如 AWS Athena 和 Google BigQuery（大数据查询），以及 Google Cloud Dataflow 和 AWS Glu（大数据转换）。基础底层基础云平台包括虚拟机（VM）、私有网络（VPC）、虚拟化块存储、身份和访问管理（IAM）以及计费和监控。

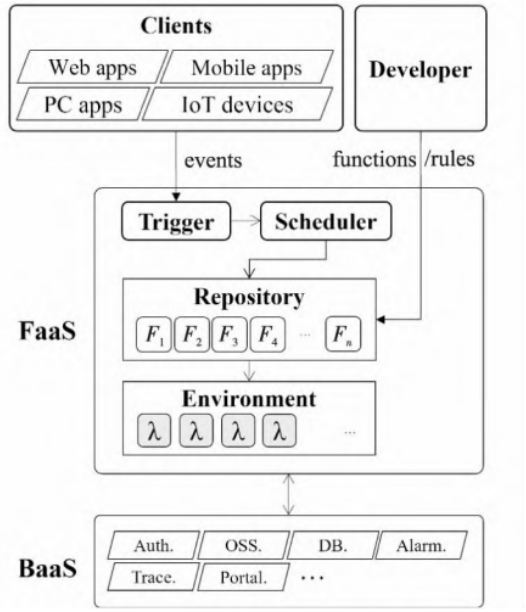

Fig. 1 无服务器计算系统架构

通过自底向上的逻辑分析，可以将无服务器计算架构解耦为四个堆栈层：虚拟化、封装、系统编排和系统协调。

虚拟化层：虚拟化层在性能和功能安全的沙盒中实现函数隔离。沙盒为应用服务代码运行时提供运行时环境、依赖项和系统库。为了防止多应用或多租户场景下的资源访问，云厂商通常采用容器/虚拟机来实现隔离。目前，流行的沙盒技术有 Docker、gVi-sor 、Kata 、Firecracker 和 Unikernel。

封装层：封装层中的各种中间件可以实现自定义的函数触发器和执行，并且它们还提供用于通信和监控的数据度量收集。这些额外的中间件称为 sidecar。它分离了服务的业务逻辑，并实现了函数和底层平台之间的松散耦合。同时，为了加快实例启动和初始化，封装层通常使用预热池。此外无服务器系统可以通过分析负载模式来使用预测，以一对一的方式预热每个函数，或者为所有函数构建一个模板，以通过一对一的方法根据运行时特性动态安装需求 (REQ)。

系统编排层：系统编排层允许用户配置触发器和绑定规则，通过随着负载变化动态调整来保证用户应用的高可用性和稳定性。通过云编排，线上线下调度相结合，可以避免资源争用，回收闲置资源，缓解共置功能的性能下降。上述实现通常也集成到容

器编排服务中（例如, Google Kubernetes 和 Docker Swarm）。而在无服务器系统中，资源监视器、控制器和负载均衡器被整合以解决调度挑战。它们使无服务器系统能够在三个不同级别上实现调度优化：分别是资源级、实例级和应用程序级。

系统协调层：系统协调层由一系列后端即服务 (BaaS) 组件组成，这些组件使用统一的 API 和 SDK 将后端服务集成到函数中。显然，它不同于使用云之外的本地物理服务的传统中间件。这些 BaaS 服务提供存储、队列服务、触发器绑定、API 网关、数据缓存、DevOps 工具等定制组件以更好地满足系统编排层的灵活性要求。

2.2 无服务器计算的特性与优势

Serverless 系统具有以下重要技术特性：

（1）自动扩展。函数实例可以从活跃节点直接扩容，以应对批量时间，具备一定的容错能力。识别无服务器系统不可或缺的因素是在适应工作负载动态时执行水平和垂直扩展。允许应用程序将实例数量缩放到零也引入了一个挑战——冷启动。当一个函数遇到冷启动时，实例需要从头开始，初始化软件环境，并加载特定于应用程序的代码。这些步骤会明显增加服务响应时间，导致影响服务质量。

（2）灵活的调度。轻量化、细粒度和短周期的函数是使 Serverless 系统简易、稳定和保持高性能的基础。由于应用程序不再绑定到特定的服务器，无服务器控制器根据集群中的资源使用情况动态调度应用程序，同时确保负载平衡和性能保证。对于更健壮和可用的无服务器系统，灵活的调度允许工作负载查询分布在更广泛的区域。它避免了在节点不可用或崩溃的情况下严重的性能下降或对服务连续性的破坏。

（3）事件驱动。触发器定义并量化事件-业务的关系，提供实时事件监听服务。无服务器应用程序由事件触发, 例如 REST-ful HTTP 查询的到达、消息队列的更新或存储服务的新数据。通过使用触发器和规则将事件绑定到函数，控制器和函数可以使用封装在上下文属性中的元数据。这使事件和系统之间的关系变得可检测，从而实现对不同事件的不同协作响应。

（4）现收现付。无服务器计费模型将计算能力的成本从资本支出转变为运营支出。这种模式消除了用户根据峰值负载购买专属服务器的需求。即用即付模型通过共享网络、磁盘、CPU、内存等资源，只指示应用实际使用的资源，无论实例是运行还是空闲。

可以看出, Serverless 技术既适用于短周期、访问量随机波动的即时服务场景，也适用于高吞吐、高并发、数据密集、算力有限和环境多样化等高要求应用场景。

对于云提供商而言，无服务器计算可促进业务增长，因为使云更易于编程有助于吸引新客户并帮助现有客户更多地使用云产品。此外，短运行时间、小内存占用和无状态特性通过使云提供商更容易找到运行这些任务的未使用资源来改善统计多路复用。云提供商还可以使用不太流行的计算机，因为实例类型取决于云提供商，例如可能对多服务器云客户没有吸引力的旧服务器。这两个好处都增加了现有资源的收入。

无服务器计算对云用户非常友好，因为新手可以在不了解云基础架构的情况下部署功能，因为专家可以节省开发时间并专注于其应用程序特有的问题。无服务器用户可以节省资金，因为这些函数仅在事件发生时执行，而细粒度计费意味着他们只需为使用的内容付费，而不是为保留的内容付费。

研究人员一直被无服务器计算所吸引，尤其是云函数，因为它是一种新的通用计算抽象，有望成为云计算的未来，并且有很多机会可以提高当前的性能并克服其当前的局限性。

2.3 无服务器计算遇到的挑战

2.3.1 缩容至零 (Scale to Zero) 问题

Serverless 系统的高吞吐特性源自函数实例快速扩容机制。然而，当访问量降至零时，系统将释放全部函数实例与计算资源。若之后请求到达而运行环境中没有对应的活跃函数实例时，系统将重新生成该函数实例，该过程称为冷启动。相比已有实例的快速扩容机制，冷启动存在着不可忽视的时延问题。

问题的关键在于延长函数实例的活跃期或采用缓存技术支持失活实例的热启动。当请求再次到达时，系统跳过检索仓库与生成实例的环节，直接从缓存中检出已有的函数实例副本，缩短响应时延。一种可行的方式是将空闲的尚未释放的容器节点和函数实例组织成队列结构，以提高缓存检索效率，减小系统时延。

2.3.2 数据调度与缓存性能瓶颈

函数运行伴随着数据传输，当数据源与函数运行环境在网络、地理空间上相距较远且数据集规模足够大时，则性能瓶颈出现在数据集的调度与传输上，造成严重的网络负载与时延。一种方法是把规模小得多的函数文件传输到数据源附近的环境中运行，从而降低网络负载。然而该方法只有函数计算结果规模远小于输入数据集时才有提升效果。该文献也遗留了分布式多数据源场景中数据传输、流式

Serverless 函数计算等未解决的问题。

引入缓存组件能有效提升数据处理性能，因此还需要研究适用于 Serverless 系统的动态分布式数据集缓存策略。当前，云提供商会为每一个运行环境提供可“就地”使用的数据缓存服务，而这种方法价格不菲，且尚未解决多函数 - 多数据集异步协作、分布式负载均衡等场景中的缓存需求。

2.3.3 触发器-调度器性能瓶颈

触发器-调度器是 Serverless 系统的重要结构，通常由 API 网关和调度队列实现，负责事件监听、任务分发等作业。对比中心化与分布式两种调度架构，前者暴露出吞吐瓶颈，而后者则存在自治节点负载不均的问题。Hivemind 实现了总体负载均衡的分布式调度架构，在并行计算单元队列的基础上设置全局负载均衡器，实时进行队列负载监控和健康检查，当队列容量饱和或异常时，当前队列中的任务可以转移到其他队列，以兼顾负载均衡和吞吐性能。

3 近年的相关研究工作

3.1 IceBreaker

IceBreaker 的核心思想在于通过异构服务器来实现 serverless 函数的 keep-warm 策略，其通过动态确定具有成本效益的节点类型来根据函数的下一次调用随时间变化的概率预热函数来实现这一点。IceBreaker 提出了一种基于傅里叶变化和一元二次多项式拟合的请求时间预测方法和判定标准 utility score，用于决定是否进行预热以及预热的方式。

通过实验可以发现，虽然低端服务器的执行效率比高端服务器低，但是低端服务器上的执行和热启动时间可能会比高端服务器上的执行和冷启动时间短。对低端服务器和高端服务器使用不同的 keep alive 策略有助于降低服务时间和开销。

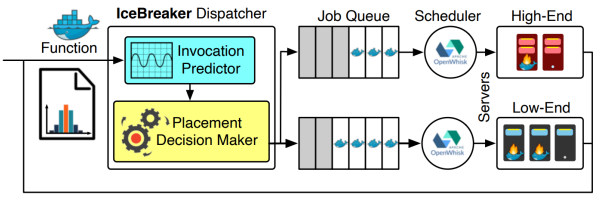

Fig. 2 IceBreaker 设计概览

IceBreaker 的设计如 2所示，它主要由两个部件组成：

1) 函数调用预测方案 (FIP)。FIP 帮助 IceBreaker 预测函数是否会在特定时间间隔内被调用。此外，它还预测调用并发性，以便可以预热适当数量的函数实例。IceBreaker 设计了一个无服务器计算特定的时间序列预测模型来对函数调用进行准确的预测。

2) 安置决策者 (PDM)。如果预测 FIP 将调用某个函数，则 PDM 会决定在何处预热函数：高端服务器或低端服务器，或者根本不预热。它根据效用分数做出此决定，该效用分数是通过考虑多个因素计算得出的，例如 FIP 的准确性和在高端服务器上预热所获得的加速。

通过采用异构性，IceBreaker 允许在相同的成本预算下使用更多的节点，从而使得更多的函数 keep warm 并减少高负载期间的等待时间。实验结果表明，IceBreaker 使用具有代表性的无服务器应用程序和行业级工作负载跟踪，将总体 keep-alive 成本降低了 45%，执行时间减少了 27% 。

3.2 HiveMind

HiveMind 是一个端到端用于 cloud-edge system 的硬件-软件系统堆栈，其核心思想是针对 serverless 和 edge computing 进行优化。HiveMind 包含一个自动进行云和边缘设备之间任务数据分配的声明式编程接口，一个基于无服务器计算的集中式控制器和一个可重构的基于 FPGA 的硬件加速结构。HiveMind 的设计如 3所示。

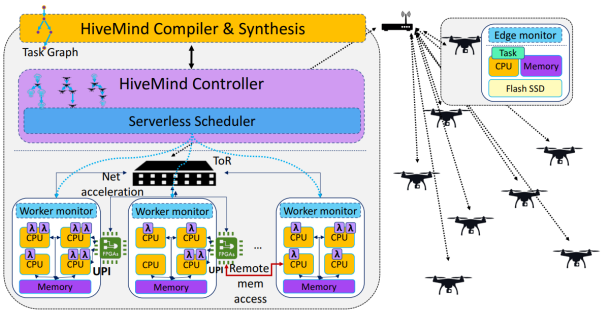

Fig. 3 Hivemind 设计概览

HiveMind 在集中式的控制器中实现了基于无服务器的云调度器，用于处理任务放置和函数实例化。HiveMind 在集群中的每台服务器上都部署了一个 worker monitor 用于定期监视性能和服务器利用率，使用这些监视器，调度程序可以识别具有足够资源的节点来执行新的任务。

调度程序实现了两项优化。首先，基于一个新的远程内存协议，函数间可以快速进行内存数据交换。调度程序会尝试将子函数与其父函数放在同一容器中，以避免通过 CouchDB 进行昂贵的数据交

换。第二个优化是减少实例化开销。HiveMind 不会立即终止空闲容器，以防新功能在不久的将来到达可以使用它。HiveMind 使用多个调度程序，每个调度程序负责一个任务子集，但对所有云和边缘资源具有全局可见性。

3.3 Molecule

Molecule 的提出主要是为了实现异构硬件的 Serverless 支持。Molecule 为无服务器应用程序启用通用设备（例如 Nvidia DPU）和特定领域加速器（例如 FPGA 和 GPU），显著提高了 Function 密度（提高 50%）和应用程序性能（高达 34.6 倍）。其设计如 4所示。

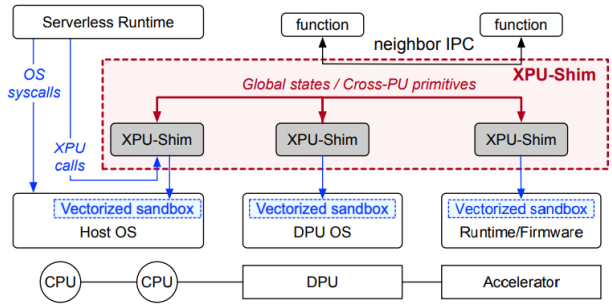

Fig. 4 Molecule 设计概览

Molecule 由两个重要部件组成：

1) XPU-Shim。XPU-Shim 是一个间接层，用于弥合底层多操作系统系统（使用通用设备时）和无服务器运行时（即 Molecule）之间的差距。XPU-Shim 依赖于两个关键原语（Distributed Capability 和 Neighbor IPC）来处理分布式硬件和操作系统。

2)Vectorized sandbox。为了支持 FPGA 的函数操作，Molecule 提出了一个通用的矢量化 sandbox，它扩展了现有的抽象以支持并发 sandbox 的创建和调用。Vectorized sandbox 依赖于 FPGA 中的包装器来保证实例之间的安全性和性能隔离。XPU-Shim 利用 Vectorized sandbox 的接口来管理异构功能。

3.4 Jukebox

Jukebox 是一个记录和重放指令预取器，用于减少预热函数实例的启动延迟。在服务器上交错执行很少调用的函数会导致调用之间每个函数的微体系结构状态发生抖动，同时函数较短的执行时间阻碍了缓存层次结构的预热延迟分摊。预热函数实例时性能损失的最大来源(平均 56%)是在 CPU front-end，特别是指令的 on-chip misses，Jukebox 的提出则是为了解决这一问题，其设计如 5所示。

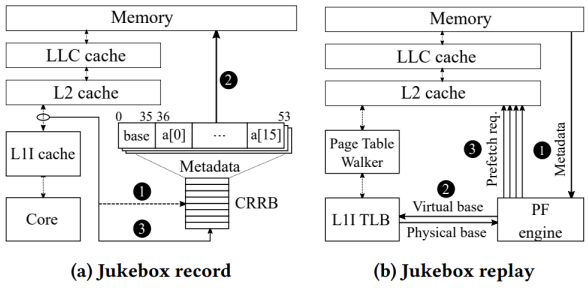

Fig. 5 Jukebox 设计概览

Jukebox 的记录过程是一个 FIFO 的记录 L2 miss 的过程。在 L1-I 未命中时，请求将像往常一样转发到 L2。在 L2 命中时，Jukebox 记录机制不采取任何操作，有效地过滤所有 L2 命中。如果 L2 中有未命中，当未命中最终返回到 L1-I 时，Jukebox 会将其记录下来。这是通过生成对 CRRB 的查找来完成的，其中根据现有条目 1 检查相应代码区域的虚拟地址。代码区域虚拟地址是通过获取与 CRRB 指针对应的丢失块虚拟地址的最高有效位（38 位）生成的。如果找到匹配的条目 n^{th}，预取器将在找到的条目的访问向量中设置位，其中 n 是缓存行在代码区域内的偏移量。否则，CRRB 中最旧的条目将被逐出 2 并分配一个新条目 3 。被逐出的条目被写入内存，可选择绕过缓存层次结构，因为元数据的 on-chip 重用不是预期的。

Jukebox 的重放阶段是由操作系统在收到一个新的函数调用时触发的。预取引擎从元数据区域按照写入内存的相同顺序读取元数据。元数据条目被预取到预取逻辑内部的一个小 FIFO 中。一旦元数据条目从内存中返回，预取器将代码区域的基址传递给 I-TLB2，像正常的代码请求一样触发地址转换。这有两个目的：首先，它确保 Jukebox 不依赖于物理地址，这些地址可能会因为正常的操作系统活动（如分页或内存压缩）而改变。其次，它有效地预先填充了 TLB 的代码页的翻译。一旦知道了代码区域的物理基址，利用条目访问向量的信息，预取引擎就会重建代码区域内每个被访问的缓存行的完整地址，并将它们排入 L2 预取队列。

实验结果表明，Jukebox 的每个函数实例仅需要 32KB 的元数据，并能将各种函数的性能平均提高 18.7%。

3.5 vHive

vHive 是一个针对无服务器实验的开源框架，它使系统研究人员能够在整个无服务器堆栈中进行创新。vHive 集成了来自领先的无服务器提

供商的开源产品级组件，即 Amazon Firecracker, Containerd, Kubernetes, and Knative，它们提供最新的虚拟化、快照和集群编排技术，以及用于功能部署和基准测试的工具链。

4 总结与展望

无服务器计算使云更易于使用，从而吸引更多能够并且将会使用它的人。它消除了当今服务器计算强加给应用程序开发人员的手动资源管理和优化的需要。无服务器计算的使用将越来越广泛。混合云本地应用程序将随着时间的推移而减少。无服务器计算仍处于起步阶段，未来几年仍然存在极大的发展潜力。

Serverless 技术相比于其他传统的云服务模式在基础设施管理、软件开发、并发性和专业性要求等方面存在诸多优势。在云计算快速发展和传统产业转型的背景下，合理选择 Serverless 作为应用开发的支撑技术将有助于业务的高效实现。

参 考 文 献

[1] Roy R B, Patel T, Tiwari D. IceBreaker: warming serverless functions better with heterogeneity[C]//Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 753-767.

[2] Patterson L, Pigorovsky D, Dempsey B, et al. HiveMind: a hardware-software system stack for serverless edge swarms[C]//Proceedings of the 49th Annual International Symposium on Computer Architecture. 2022: 800-816.

[3] Du D, Liu Q, Jiang X, et al. Serverless computing on heterogeneous computers[C]//Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2022: 797-813.

[4] Schall D, Margaritov A, Ustiugov D, et al. Lukewarm serverless functions: characterization and optimization[C]//ISCA. 2022: 757-770.。

[5] Ustiugov D, Petrov P, Kogias M, et al. Benchmarking, analysis, and optimization of serverless function snapshots[C]//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. 2021: 559-572.

[6] 杨柏蔼, 赵山, 刘芳. 无服务器计算技术研究综述 [J]. 计算机工程与科学, 2022.