华 中 科 技 大 学

研究生课程考试答题本

考 生 姓 名 王隆

考 生 学 号 M202273788

系 、 年 级 计算机系 2022

班 级 硕 2207 班

考 试 科 目 数据中心技术

考 试 日 期 2022-12-25

评　　分

题 号	得 分	题 号	得 分
1		9	
2		1 0	
3		1 1	
4		1 2	
5		1 3	
6		1 4	
7		1 5	
8			

总 分 ：	评 卷 人 ：

注：1.无评卷人签名试卷无效。

2.必须用钢笔或圆珠笔阅卷，使用红色，用铅笔阅卷无效。

摘　要

在过去十年中，无服务器计算作为一个新兴的领域收到了越来越多的关注，因为它在降低成本、减少延迟、提高可扩展性等方面表现出巨大优势。云计算实在软件与硬件虚拟化之后出现的技术，具有良好的可扩展性、灵活性，用户可以直接通过互联网来访问这些云服务。云计算根据提供的服务可以分为软件即服务(SaaS)、平台即服务(PaaS)、基础设施即服务(IaaS)，然而，对于这些云服务的管理是一个挑战。云服务的管理需要从多方面考虑：可用性、负载均衡、自动缩放、安全性等等，这些指标对于优秀的云服务而言缺一不可，而兼顾考虑这些参数有着相当的难度。

这些挑战引入了另一种云计算的模型，称为无服务器云计算。无服务器云计算包含后端即服务(BaaS)与功能即服务(FaaS)，无服务器模型让开发人员专注于应用程序逻辑，而不是服务器端的管理和配置。无服务器云计算有许多突出的有点，其中之一就是可扩展性，在无服务器计算中，应用程序会自动按需扩展和缩减，开发人员不必担心扩展问题。

本综述首先介绍无服务器的相关概念，然后讨论适合部署在无服务器上的应用程序，最后引出无服务器的性能与安全问题。

关键词：云计算；无服务器计算；

Abstract

In the past decade, serverless computing has received more and more attention as an emerging field, because it has shown great advantages in reducing costs, reducing latency, and improving scalability. Cloud computing is a technology that emerged after software and hardware virtualization. It has good scalability and flexibility. Users can directly access these cloud services through the Internet. Cloud computing can be divided into software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) according to the services provided. However, the management of these cloud services is a challenge. The management of cloud services needs to be considered from many aspects: availability, load balancing, auto-scaling, security, etc. These indicators are indispensable for excellent cloud services, and it is quite difficult to take these parameters into consideration.

These challenges have introduced another cloud computing model called serverless cloud computing. Serverless cloud computing includes backend as a service (BaaS) and function as a service (FaaS). The serverless model allows developers to focus on application logic rather than server-side management and configuration. One of the many advantages of serverless cloud computing is scalability. In serverless computing, applications are automatically scaled up and down on demand, and developers don't have to worry about scaling.

This overview first introduces the concepts of serverless, then discusses applications suitable for deployment on serverless, and finally leads to serverless performance and security issues.

Keywords: cloud computing, serverless computing

目　录

1 绪 论

1.1 背景介绍

云计算出现在软件和硬件基础设施的虚拟化之后，云提供商越来越多地采用它来为客户提供服务。客户可以通过互联网访问这些云服务，极大地便利了服务对于用户的提供。许多软件开发人员也倾向于使用云技术解决问题，因为云技术本身具有可扩展性、可用性和灵活性等优势。一般来说，云计算根据提供的服务分为三大类，即软件即服务(SaaS)、平台即服(PaaS)和基础设施即服务(IaaS)。 在 SaaS 中，云提供商将不同类型的软件作为服务提供给用户，在这种类型的云服务中，用户不负责服务的开发、部署和管理，只管使用而不用担心它们的设置、配置等。在 PaaS 中，云公司提供网络接入、存储、服务器、操作系统等服务，供开发者购买，这些服务更多地面向开发人员。开发人员访问这些服务来部署、运行和管理他们自己的应用程序。在这种云服务中，开发人员需要负责部署和管理他们的软件以确保应用程序运行，但他们不控制服务。 最后，在 IaaS 类别中，云消费者需要控制和管理网络访问、服务器、操作系统和存储等多种服务。而管理云服务本身就是一个挑战。例如其中可能涉及到可用性、负载平衡、自动缩放、安全性、监控等工作。

这些挑战催生了另一种云计算模型，称为无服务器云计算。无服务器云计算提供后端即服务(BaaS)和功能即服务(FaaS)。BaaS 包括存储、消息传递、用户管理等服务。而 FaaS 被认为是无服务器的最主要模型，也被称为“事件驱动功能”，FaaS 依赖于 BaaS 提供的服务，例如数据库、消息传递、用户身份验证等。无服务器云计算在现有的云计算范式上增加了一个额外的抽象层，无服务器模型让开发人员专注于应用程序逻辑，而不是服务器端的管理和配置。无服务器云计算有很多好的特性，其中之一就是可扩展性。在无服务器计算中，应用程序会自动按需扩展和缩减，开发人员不必担心扩展问题。无服务器计算的另一个特点是按资源使用量付费。这种云计算范式根据实际资源使用情况向开发人员收费。例如，在应用程序空闲的情况下部署应用程序不会对开发人员产生成本，而无服务器提供商只会在应用程序开始使用资源时收费。无服务器云计算也有着一些缺点，首

先，无服务器函数的短运行特性限制了应用范围；其次，无服务器云计算将应用程序分解为功能会引入性能开销；第三，就安全属性而言，传统的执行抽象如 VM 和容器并不是容纳来自不同客户的这些小功能的最佳选择。

1.2 无服务器模型

无服务器模型多数时候指功能即服务(FaaS)，客户可以自由开发、运行和管理其应用程序的功能，而无需在开发和启动应用程序时通常构建和维护基础设施的复杂性。无服务器服务应满足以下要求：

功能级管理。无服务器中的基本单元是功能。所有功能都是相互独立的，即使它们实际上来自同一个应用程序，在无服务平台中部署、调度、隔离的单元都是函数。

短期运行。由于与整个应用程序相比，功能通常是较小的单元，因此预计它们会在短时间内完成。但是，随着无服务编程模型越来越流行，Serverless 上的工作负载现在各不相同，大数据分析功能不一定很快完成。因此，未来可能会放宽对最长执行时间的限制。

透明度。用户的执行环境对于无服务器来说是透明的。透明度对于无服务器至关重要，因为它将用户从代码部署和管理的负担中解放出来。

无状态。函数是无状态的，仅描述任务处理的应用程序逻辑。所有必需的状态都是从外部存储导入的，函数的无状态特性使无服务器对崩溃等异常时间更具有健壮性，因为无服务器不需要在崩溃的函数中恢复状态。

付费方式。无服务器有着固定的计费方式，即云提供商只在上传的功能真正执行时才收费。这相较于按时租用的虚拟机有着更小的成本开销，因此，无服务器对云用户非常有吸引力，因为它意味着成本更低。

2 无服务平台上的应用

并非所有应用都能在无服务平台上运行，事实上，由于目前 Serverless 平台的局限性（如启动延迟大、函数间通信性能慢），只有少数应用成功移植到无服务平台上。能够移植到无服务平台的应用有着一些特点：

规模较大并且并行性较强：无服务器计算的第一个典型用例是构建一个由数千种主要独立功能组成的应用程序。在这个用例中，每个函数都包含所有必要的代码和数据，这意味着它不需要与其他函数通信。这样的需求刚好绕开了无服务平台的局限性，因此，此类应用可以通过创建数千个 worker 来轻松提升，以充分享受平台的自动扩展支持。

事件驱动的处理程序：这种类型的应用程序是等待特定类型事件的被动处理程序。一旦其监听的事件触发，处理程序就会被唤醒，开始进行相应的业务逻辑。Web 和移动应用程序 API 服务是无服务器计算的两个最常见的用例。API 服务器通常不需要长时间运行的服务器保持运行并等待用户请求，如果 API 请求数量少，那么大部分服务器计算时间都浪费在循环或者休眠等待上。通过使用无服务器，这些 API 服务不需要保存在长期运行的服务器中，可以节省成本开销。

通用任务型应用：随着无服务的快速发展，研究人员开始探索更复杂的应用程序。这些应用程序设计软件编译、单元测试和视频编码等等。尽管其中一些已得到一些无服务器平台的支持，但这些平台是特定于应用程序的，因此不足以支持各种基于任务的应用程序。Fouladi 等人[1]提出了一种中间表示来将基于任务的通用应用程序与无服务器平台连接起来，要求开发人员将应用程序划分为 thunk，以便每个 thunk 将被翻译成函数并在不同的容器中执行。实验结果表明，使用该方式的无服务器平台的执行时间与传统的多核执行相当，这表明无服务器架构在有效支持各种应用程序方面具有巨大潜力。

3 无服务器的性能

无服务器已成为一种新的计算范式，其将应用程序拆分为多个短期运行的按需函数以进行单独处理，这些应用程序分解为更小的单元以实现更好的可移植性和可维护性，但这也引入了性能问题。本小节讨论无服务器框架存在的启动开销大以及由于隔离产生的通信开销的问题。首先，无服务器框架要启动一个函数，首先需要准备运行时环境，其时间成本有时甚至会超过函数本身的执行时间，这显然是不可接受的。而且对于高级语言的应用程序来说，这样的准备时间进一步加剧了；其次，由于现在工作的函数彼此分离，以前正常的函数调用现在需要考虑进程间通信和远程网络调用，这又引入了通信开销。

主流的无服务框架为了实现故障隔离和可移植性，主要是将功能部署到容器中。然而，启动一个容器是相当耗时的：从一开始就需要 100 多毫秒来初始化一个容器。 相比之下，函数的执行时间通常只需要几毫秒。因此，容器的启动开销成为无服务框架场景的一大瓶颈。为了解决容器遇到问题，研究者给出了一些解决方案：

容器缓存：引导问题的一个直接解决方案是容器缓存。其思想为：运行环境的重用。当一个功能完成时，无服务器框架可以保留其运行时环境，以便来自同一用户的相同功能可以重用它。主流的商业无服务器平台都采用了这种机制，不太可能为新功能启动新容器。

预热：容器缓存主要是为了在函数被重复调用时复用容器而设计的，但是，对于来自不同用户的相同功能，由于安全问题，容器不能被复用，此类用户仍然可能遭受启动缓慢的困扰。另一项工作引入了预热容器，为具有相似特征的函数准备运行时环境[2]。这些容器通常是在没有用户信息的情况下产生的，因此用户可以直接利用它们而不必担心他们的隐私。由于预热容器使用相同的工作流程进行初始化，因此它们的内存状态非常相似,一些研究者提出了一种基于分叉的方案来减少预热容器的内存占用[3]。当需要一个新的容器时，它将从一个预热的容器中分离出来，容器的内存页也是共享的，只有在发生写操作时才变为私有。该解决方案尤其适用于在高级语言运行时之上运行的函数，因为运行时的初始化在不同的运行中几乎相同。

容器优化：启动容器的成本不可接受，因此有研究者试图根据无服务器功能的特点来优化启动时间。Mohan 等人[4]关注大量容器网络创建的可扩展性问题，并利用网络预创建来解决该问题，还为容器提供了比普通容器更快的启动时间。

寻找其他抽象：由于容器中的启动时间较慢，无服务器框架还寻找可替代容器来隔离功能。例如，与容器相比，虚拟机可以提供更强的隔离，但它们通常具有更长的启动时间。无服务器还有许多替代抽象，例如 Google gVisor、AWS FireCracker、Unikernels 等。我们预计会有更多抽象作为工作负载在无服务器中变得更加多样化和关键。

由于无服务器框架将应用程序分解为单独的功能，现在正常的功能调用变成了通过网络进行远程调用，这带来了相当大的开销。无服务器平台忽略了功能之间的关系，因此功能可能驻留在不同的服务器上，在前面的函数退出之前，它将其输出数据存储到存储服务器，以便后续函数可以检索它作为输入。由于这两个函数和存储服务器位于三台不同的机器上，函数调用速度明显变慢，因为它涉及至少两次网络往返。为了优化功能之间的通信，研究者着眼于优化存储服务器的性能或优化整个通信路径。

优化存储服务器：主流的无服务器框架直接重用云存储服务来存储功能间的中间数据。但是，这些存储服务并不是为短期运行的功能而设计的，它们成为性能瓶颈。这个问题在无服务器中更加严重，因为它往往需要在函数之间传输大量中间数据。存储服务应该具有高度弹性，以满足无服务器分析的高 IOPS 和吞吐量要求。Pu 等人[5]还结合了不同类型的存储设备，以实现无服务器分析的性能和成本效益。Stuedi 等人[6]也是为存储和交换大规模临时数据而构建的，但它的目标是高带宽和低延迟。Zhang 等人[7]进一步提出了存储功能，将无服务器计算转移到存储服务器上，以减少数据交换开销。

优化交流路径：无服务器框架通常不关心或不关心函数之间的通信。 因此，即使两个函数恰好被安排在同一台服务器上，现有的框架仍然利用存储服务器进行数据交换。Akkus 等人[8]将专门处理这些链并将它们放在同一台机器上。此外，它还提出了分层消息队列，即在同一机器上的函数链内的本地消息将直接传递，而无需存储服务器。协同定位机制也可能有助于无服务器分析，其中应用程序可以表示为有向无环图 (DAG)。

4 无服务器的机会与挑战

首先，无服务使云用户使用高级编程语言以功能为单位组织应用程序，这意味着迁移单元不再是 VM，而是更小的无服务功能模块。较小的功能可以轻松地在云之间迁移，如果迁移函数的开销仍然很大，无服务器函数的无状态属性允许我们在新云上重新启动一个新实例并中止旧实例。其次，今天的无服务器平台支持短期运行的功能，大多数功能必须在几分钟内完成。通过使用无服务器计算实现联合云模型，用户可以将功能部署在不同的云上，同时享受最低的价格和最好的性能。例如，假设有一个云在没有快速加速器的情况下提供廉价的无服务器服务，而另一个云拥有加速器但提供更昂贵的无服务器服务，我们可以将大部分计算功能部署在第一云，将加速器相关功能上传到第二云以节省开销。

但是，在无服务器模型中也存在一些挑战。首先，当今无服务器云中使用的定义和 API 彼此不同，即使其他云也有类似的服务，但这些服务有不同的 API，这阻碍了功能在云之间的切换或迁移；其次，选出最适合用户的云平台是一项挑战。云的选择取决于多种因素，例如云与用户之间的地理距离、每个云的动态定价和服务质量，以及云之间的通信效率，不明智的选择可能会导致位于不同云的功能之间的通信增加；第三，云服务质量的动态变化仍然难以处理，跨云迁移现有功能实例可能并不比在目标云中重新启动一个新功能实例更好，尽管重启策略非常适合通常无状态的无服务器计算，但重启可能仍然涉及数据库之间的持久状态迁移，因此，需要一个性能模型来估算迁移和重启的开销，从而做出最具成本效益的选择。

参考文献

[1] Fouladi S, Romero F, Iter D, et al. From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019: 475-488.

[2] Oakes E, Yang L, Zhou D, et al. {SOCK}: Rapid task provisioning with {Serverless-Optimized} containers. In 2018 USENIX Annual Technical Conference (USENIX ATC 18). 2018: 57-70.

[3] Du D, Yu T, Xia Y, et al. Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020: 467-481.

[4] Mohan A, Sane H, Doshi K, et al. Agile cold starts for scalable serverless. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19). 2019.

[5] Pu Q, Venkataraman S, Stoica I. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). 2019: 193-206.

[6] Stuedi P, Trivedi A, Pfefferle J, et al. Unification of Temporary Storage in the {NodeKernel} Architecture. In 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019: 767-782.

[7] Zhang T, Xie D, Li F, et al. Narrowing the gap between serverless and its state with storage functions. In Proceedings of the ACM Symposium on Cloud Computing. 2019: 1-12.

[8] Akkus I E, Chen R, Rimac I, et al. {SAND}: Towards {High-Performance} Serverless Computing. In 2018 Usenix Annual Technical Conference (USENIX ATC 18). 2018: 923-935.