译Process-as-a-Service-FaaSt-Stateful-Computing-with-Optimized-Data-Planes

题目

Process-as-a-Service: FaaSt Stateful Computing with Optimized Data Planes

进程作为服务 (PraaS)： 用优化的数据平面进行快速的有状态计算

摘要

细粒度和短暂的函数为许多新应用程序提供动力，这些应用程序受益于 serverless 平台的弹性扩展和较低的计算成本。但是，它们受到昂贵且有限的通信以及高调用延迟的阻碍。函数无法跨调用跟踪状态，必须依赖远程存储，使得工作流和应用程序具有任务之间的依赖关系，或者依赖于 worker 之间的通信，很难使用函数即服务 (FaaS) 计算来实现。

为了继续 serverless 革命，我们引入了进程即服务 (PraaS) 的概念，并展示了如何调整已建立的操作系统抽象来建模和实现动态配置的云计算工作者。我们展示了新的 serverless 数据平面，可将调用性能提高多达 32 倍，同时保留 serverless 工作者的短暂和弹性特性。最后，我们为 serverless 构建了一个统一且可移植的通信接口，实现了优化的点对点通信，允许 worker 并行解决问题。以节省成本。

1. 引言

在不到十年的时间里，函数即服务 (FaaS) 已将自己确立为云中的基本编程模型之一。在 FaaS 中，用户调用短期运行和无状态函数并受益于按需付费和降低成本，而云提供商获得更有效的资源使用和重用闲置硬件的机会 [1-3]。 serverless 函数是 FaaS 所基于的计算模型，已用于 Web 应用程序、媒体处理、数据分析、机器学习和科学计算 [4-9]。尽管 FaaS 在降低无状态计算和微服务的成本方面取得了显著成功，但 serverless 计算的广泛采用受到其执行模型的限制 [4, 10-12] 的阻碍。

作为 serverless 系统中趋势工作负载的一个示例，我们考虑分布式机器学习解决方案 [4, 13-16]。在那里，每个调用的函数都必须使用数据子集计算新的梯度。虽然 FaaS 的无状态特性简化了部署和资源管理，但组合新的权重并在下一轮调用中使用它们会产生重大的性能开销，因为在每一轮更新中，必须从外部云存储中写入和读取输出，并且每个新的 round 需要通过整个云控制平面的调用。

如果没有直接的通信渠道，像 serverless 联合机器学习这样的服务难以实施且效率低下。临时函数 [10, 12] 中缺乏状态管理可以部分解决，例如基于存储的共享状态、事务和日志 [17-20]。然而，这些解决方案依赖于手动管理的非 serverless 存储实例，它们会对 serverless 系统的执行时

间、吞吐量和成本产生负面影响 [12、20]。通过利用本地可用内存而不是不断需要访问远程、缓慢的存储，并且能够绕过控制平面进行重复调用，可以极大地提高此类系统的性能。

我们在 serverless 世界中提出了一种新范式：进程即服务 (*PraaS*)——我们无缝地整合状态，保留 FaaS 计算的短暂性，并彻底改革 serverless 系统架构。*PraaS* 深受经典操作系统 (OS) 设计的启发，并将经受住时间考验的概念转移到粒状云计算环境中。具体来说，这种新模型使我们能够解决当前 FaaS 系统的三个最重要的限制：缺乏一致和低延迟的状态，每次调用都涉及复杂的控制平面，以及缺乏统一和可移植的通信接口。

	IaaS	CaaS	PraaS	FaaS
Computation Unit	Virtual machine	Container	Process	Function
External Interface	SSH, TCP, HTTP, RPC, RDMA	SSH, TCP, HTTP, RPC	HTTP, TCP	HTTP
Lifetime	Months	Days, hours	Minutes, hours	Seconds
State Duration	Persistent	Transient	Transient	Ephemeral
State Location	Local disk, memory	Memory, cloud storage	Memory, cloud storage	Cloud storage
Provisioning	Manual, minutes	Semi-automatic, secs	Automatic, msecs	Automatic, msecs
Compute Resources	Persistent	Persistent	Ephemeral	Ephemeral
Billing	Provisioned	Provisioned	Pay-as-you-go	Pay-as-you-go

Figure 1: Evolution of computing platforms in the cloud - *PraaS* enables **state persistence** for **ephemeral workers.**

PraaS 是云计算发展的一个新步骤，它在持久分配的性能和临时 worker 的弹性之间提供了更好的平衡（图 1）。它引入了会话的概念，即一个跨调用保留本地状态的执行环境，并公开了一个通信通道，以便用户可以直接发送数据和请求。

进程打开了直接热函数调用的可能性。在传统的 FaaS 平台中，每次调用都会遍历控制平面中的集中式（和多步）分配和路由逻辑（第 2.1 节）。此外， serverless 函数大多是短期运行的 [21]，并且快速网络传输的日益增加的可用性对优化每次调用中涉及的 serverless 控制平面造成了进一步的压力。在 *PraaS* 中，会话公开的通信通道允许我们公开与会话中执行的函数的直接连接。因此，我们提出了一个进程数据平面，它通过与函数沙箱的直接连接提供低延迟的暖执行。我们没有应用优化来减少控制开销，而是通过将调用有效负载直接提交到进程实例来完全从数据路径中移除控制平面开销 [22]。我们提出的新数据平面支持使用 serverless 函数进行并行和粒度计算 [23, 24]，其中调用开销的相对影响非常高。

PraaS 的另一个好处是能够在单个进程内的会话之间直接移动数据。这允许用户部署大规模应用程序，例如可以通过访问状态内存直接进行通信的分布式 ML。到目前为止，函数间通信一直受到限制，迫使用户在云存储 [4、8] 之上实施标准通信模式，并引入了额外的成本、不可忽略的延迟，或者需要非 serverless 甚至手动管理的存储服务。受分区全局地址空间 (PGAS) 的启发，我们为 serverless 函数提出了一个全局内存模型，该模型由可靠和持久的存储支持。我们为 serverless 提供了一种通信模型，它将直接消息传递的要求调整为临时函数，并在可移植和抽象的接口中实现快速通信，有效地取代了控制平面提供的传统函数：使用 *PraaS* 的直接调用比使用 AWS Lambda 快 32 倍, 适用于小型和大型有效载荷大小。

总而言之，我们做出以下贡献：

- 为有状态 FaaS 函数提供高效模型的 serverless 进程的新概念。
- 优化的 serverless 数据平面，可快速直接调用可扩展工作流和高性能 FaaS。
- 用于 serverless 函数的全局内存模型，允许优化函数间通信，并提供跨系统和云提供商的可移植接口。

2. 动机

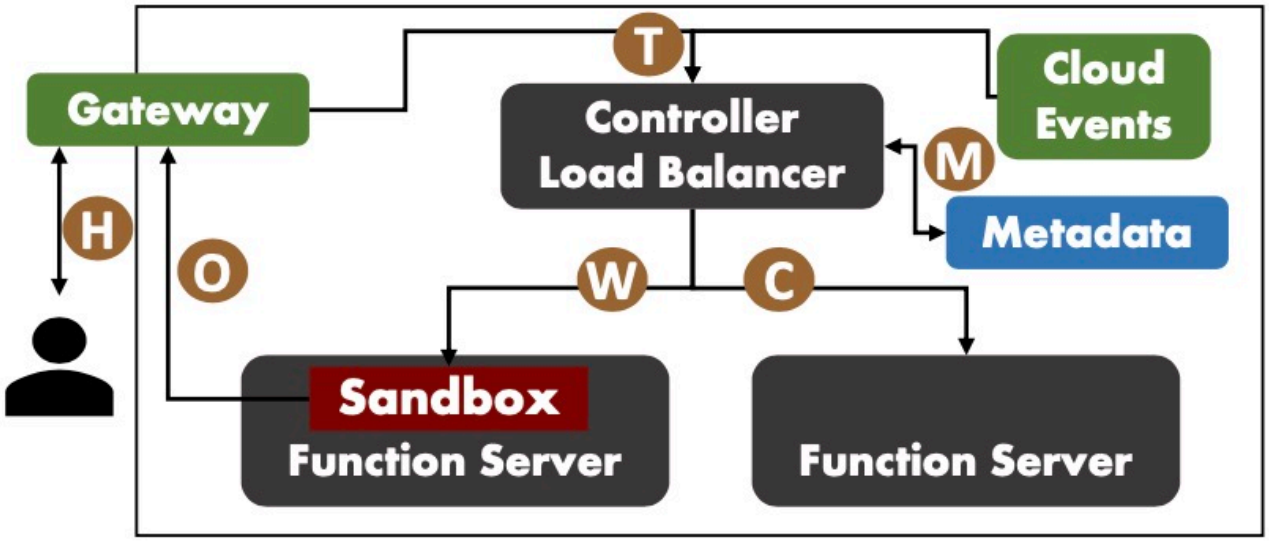


Figure 2: **FaaS control plane:** each invocation includes the management overhead (M).

函数即服务 (FaaS) 增强的云具有细粒度和弹性的编程模型，并且 FaaS 已经找到了进入主要云系统的途径 [25-28]。函数是无状态的，调用不能依赖于先前执行产生的资源和数据。函数实例不是使用持久的和用户控制的虚拟机和容器，而是动态地放置在云管理的沙箱中，例如容器和轻量级虚拟机 [29]。冷调用需要分配一个新的沙箱，这会增加调用延迟的大量开销 [11, 30]。连续的热调用可以受益于通过先前的冷调用重用沙箱，从而实现更低的调用延迟。因此，云系统尝试在调用后保留虚拟机和容器，采用复杂而复杂的保留和预热策略 [31-34]，并以更高的内存消耗换取更快的执行速度。此外，灵活的即用即付计费是 serverless 系统的另一个显着优势：用户只需为计算时间和使用的资源付费。然而，除了 serverless 计算提供的好处之外，它也有一些缺点，例如由于复杂的控制平面导致的延迟较高、由于不存在本地状态而导致的数据局部性差以及通信成本高。[10、11、35、36]。

2.1 Serverless 控制平面

Serverless 执行模型要求函数执行器是动态分配的，用户不配置云资源。在实践中，大多数现代 FaaS 系统将其实现为集中管理和路由系统，将用户数据移动到函数实例。图 2 展示了 serverless 控制平面的高级概述。它包括抽象的 REST API 和具有持久网络地址的网关，并使用

HTTP 连接 (H) 隐藏函数执行器的动态选择和分配。云事件和网关请求触发 (T) 函数调用。该函数的输入被转发到中央管理（M）负责授权、资源分配和路由执行到选定的服务器。在 AWS Lambda 中，控制逻辑负责授权请求、管理沙箱实例并将执行置于云服务器上 [29]。在 OpenWhisk [37] 中，函数执行的关键路径更长。每个调用都包括一个前端 Web 服务器、控制器、数据库、负载平衡器和一个消息队列 [38]。最后，函数的数据移动到暖 (W) 或冷 (C) 沙箱。执行完成后，函数的输出通过网关（O）返回给用户。

控制逻辑的许多步骤为每次调用增加了两位数的毫秒延迟，并且需要多次复制用户有效负载，即使 serverless 系统优化冷启动，并且后续调用在可用时重用相同的暖沙箱。控制平面的开销可以支配执行时间，并且远高于传输输入参数所需的网络传输时间[11]。漫长而复杂的调用路径在分布式应用程序和 serverless 工作流中更加明显，这些工作流多次调用具有大量负载的函数。serverless 工作流的优化工作被引导到重用函数实例和提供专用的有状态执行器 [39-42]，但没有提供可以推广的解决方案。替代方法包括分散调度和直接分配 [3, 39]，但它们仅限于针对 serverless 工作流和 RDMA 加速优化的系统。

观察：每个 serverless 调用都会产生身份验证、资源管理和跨云服务数据复制的开销。但是，在许多热调用中，函数放置并没有改变，因此不需要控制逻辑。

2.2 Serverless 状态

Storage Type	1B	1 MB	10 MB
Persistent storage (AWS S3)	10.3	20.4	102.4
Key-value storage (AWS DynamoDB)	34	n/a	n/a
In-memory cache (AWS ElastiCache)	0.9	24.6	84.6

Table 1: Access time [ms] to remote cloud storage from an AWS Lambda Function with 2 GiB RAM.

函数的无状态特性使云提供商可以轻松安排它们，但对 FaaS 系统的可编程性有很大限制。计算资源分配有短暂的内存存储，不能保证在调用中持续存在。由于许多应用程序需要在调用之间保留状态，有状态函数将它们的状态放置在远程云存储中 [18-20]。虽然函数的状态位于远离计算资源的存储中，但获取和更新状态会为执行增加数十毫秒的延迟（表 1），从而导致显着的性能开销 [19, 20]。

依靠远程存储来实现有状态的函数需要每次调用都会产生存储开销，即使函数沙箱保持温暖也是如此。使用当前的 FaaS 模型，无法克服来自数据路径的远程存储访问。

观察：将 serverless 计算限制为无状态函数会给用户应用程序带来不必要的限制，从而防止数据局部性。有状态函数有助于缓解其中一些挑战，但在当前的 FaaS 模型中阻碍了它们的有效实

施。

2.3 Serverless 通信

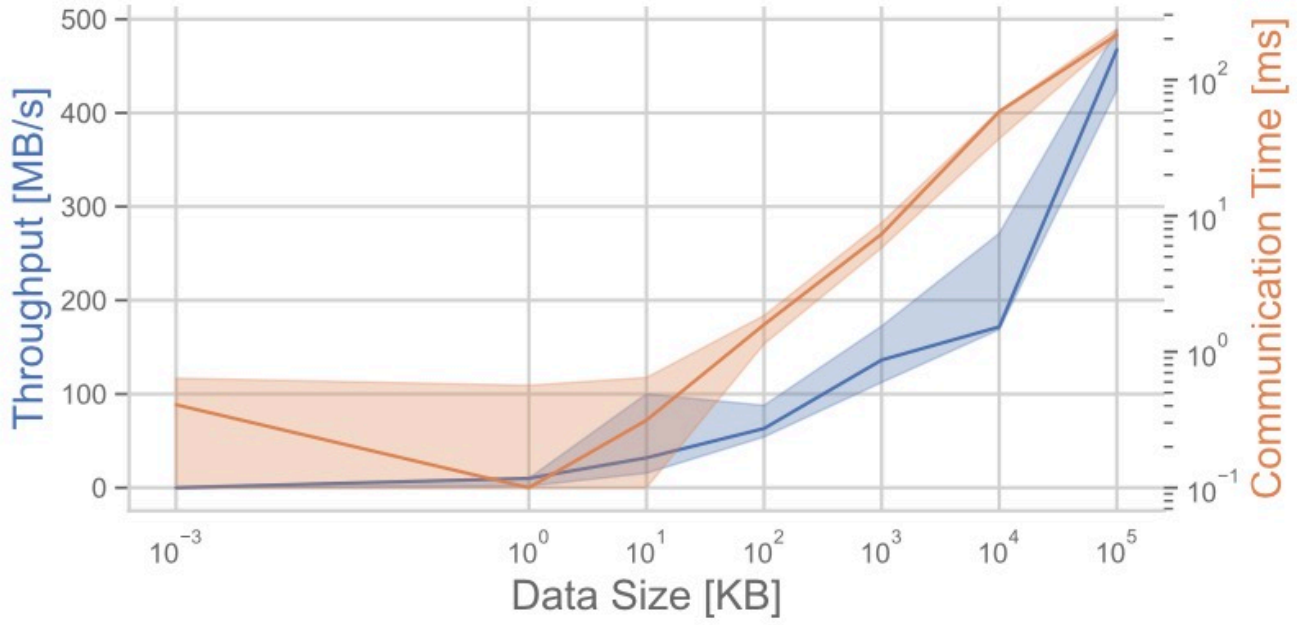


Figure 3: Latency and throughput for peer-to-peer TCP communication between Lambda functions.

FaaS 中的通信一直受到限制。消息传递范式不能直接应用于临时函数，因为没有定义 worker 的生命周期，并且函数需要邮箱来存储和访问消息。不幸的是，具有持久云存储的广泛使用的通信模式增加了不平凡的延迟和成本[10]。虽然函数之间的直接网络通信可以缓解性能问题（图 3），但函数不提供在具有动态组成员资格的函数组上实现通信所需的抽象。此外，在典型的 FaaS 部署中，函数在 NAT 之后运行并且不能接受传入连接。必须应用 UDP 和 TCP 打孔 [43, 44] 来建立函数实例之间的通信。这不仅是一个复杂的过程，而且只能在两个活动函数之间创建连接——FaaS 的编程模型不提供可靠发送消息所需的邮箱。

观察：构建并行和分布式应用程序需要廉价且可扩展的通信，但 serverless 编程模型缺乏有效的通信结构和接口。

总结

总结 FaaS 不仅仅是一个用于不规则和轻量级工作负载的平台，而且 serverless 编程模型很好地满足了大规模并行和粒度计算的需求 [23, 24]。然而， serverless 系统必须首先克服关键限制：每次调用都涉及复杂的控制平面、缺乏快速且廉价的函数状态以及昂贵的基于存储的通信。

3. *PraaS*: 进程作为服务

PraaS Concept	Inspiration
Process	POSIX process model.
Function	Thread in a process.
State	POSIX process memory.
Session Persistence	Swapping memory pages.
Communication channel	Socket in a process.
Communication model	Remote memory access, TCP
Data plane	Network data plane in Arrakis [22], kernel bypass in RDMA.

Table 2: In *PraaS*, we show how the major concepts of systems design can be used to lift the process model into a distributed and serverless space.

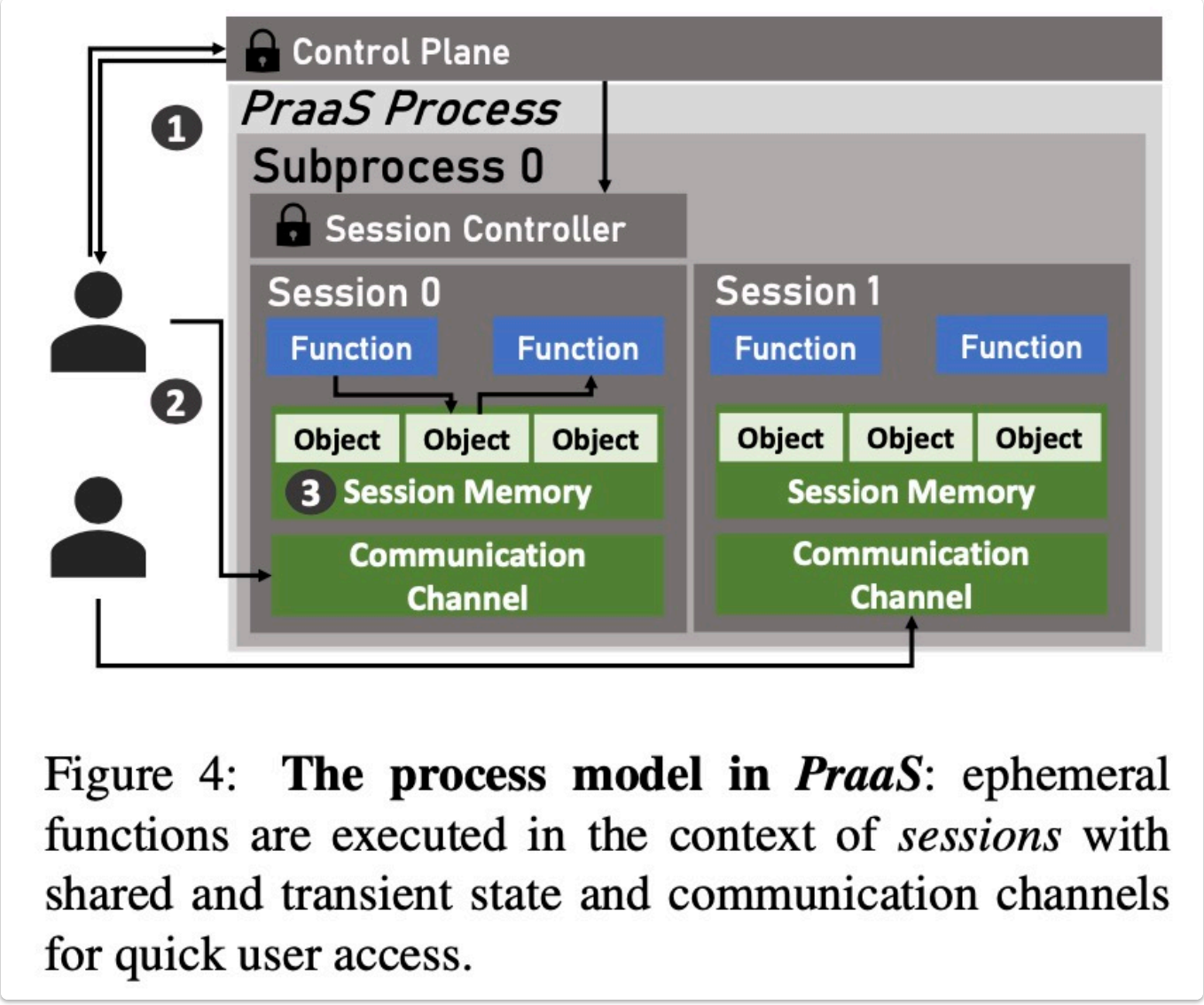


Figure 4: **The process model in *PraaS***: ephemeral functions are executed in the context of *sessions* with shared and transient state and communication channels for quick user access.

在本节中，我们将进程作为服务模型从核心设计概述开始，然后提供其特性和组件的详细信息。进程（图 4）是具有解耦和暂态的短暂函数的自然扩展。一个进程可以支持多个用户，必要时相互隔离，与一个租户相关联。*PraaS* 中的多个会话可以使用全局可访问的进程内存进行协作，使用第 4.1 节中描述的方法。

PraaS 函数实现了从 FaaS 系统已知的临时工作者的语义。我们的目标是允许快速连接并提供一种简单的方法来管理函数的状态。为了实现这个目标，每个进程函数调用都与一个进程会话相关联。在进程内分配进程会话时，会为它们分配唯一标识符、通信通道和用户定义的内存量。在会话中托管第一次调用之后，将通信通道返回给用户（1）。随后的调用可以绕过控制平面并使用 *PraaS* 数据平面 (2)。这允许函数使用会话状态中先前函数调用存储的数据，并与在同一会话中运行的其他并发函数进行通信（3）。

3.1 会话和状态

在 FaaS 中，为了提高性能，目标是保留容器并减少冷启动的频率。同样，在 *PraaS* 中，我们通过基于会话的直接调用实现机会优化，同时允许云提供商透明地扩展资源。会话包含一个唯一标识符、用于将调用路由到已经温暖的容器的通信通道，以及函数可用的暂态内存。保证在会话上

下文中调用的函数可以本地访问此内存，从而确保快速访问。因此，会话内存在创建时是静态分配的。会话可以根据需要动态分配额外的进程内存（第 4.1 节）。

隔离和共享

会话允许 *PraaS* 在单个进程实例中处理与不同用户和工作流相关的工作负载。每个会话都有自己的私有内存，与其他会话隔离（每个会话都有自己的页表）。当一个函数被调用时，它只能访问其会话内的内存，如图 4 所示 - 因此会话内的函数可以共享内存和协作。用户负责避免数据竞争（例如锁定）。函数不能直接访问不同会话的内存，但可以通过使用全局进程内存与其他会话中的函数协作，我们在第 4.1 节中详细介绍了这一点。

会话可以同时处理多个调用，但受限于它们运行的系统的内存和计算限制。云提供商可以限制单个会话可以启动的同时调用的数量，例如与会话分配的内存有关。为了允许更大的工作流，我们在 4.1 节中讨论了一个进程中多个会话的可扩展管理和通信。

示例：我们考虑使用 FaaS 的广泛使用模式，其中持久性和基于 IaaS 的应用程序将计算工作负载卸载到廉价且弹性的 serverless 加速器 [3, 45]。用户可以使用同一个会话来调用函数，这些函数将访问同一个共享内存池。*PraaS* 中的会话模型允许连续或并发调用始终访问暖和初始化应用程序的状态。此外，会话可以增强数据共享，例如，跨许多推理函数重用大型机器学习模型，允许本地通信并减少管理开销。

交换

会话不是持久的：它们的生命周期是有限的，云提供商可以随时删除它们。但是，会话是虚拟化的：它们的内存在单个会话中的调用之间共享，并且沙箱（容器或虚拟机）被删除时它也不会消失。

相反，会话被换出到持久云存储 - 标准策略是在一段时间不活动后交换会话。在一个进程中，选择不同的会话允许用户将他们的私有数据与其他用户隔离，即使属于同一个租户。

此外，当一个进程有活动会话时，它就被认为是活动的，我们不会驱逐一个进程，直到它的所有会话都被换出。会话提供了一种定义进程生命周期和选择进程容器以进行驱逐的自然方式。默认情况下，当会话控制器将会话换出到存储时，所有当前状态信息都将写入到唯一会话 ID 下的持久云存储中。如果不再需要会话，用户还可以完全删除会话以及任何相关的状态信息。

用户可以选择创建用户定义的检查点处理程序，用于定义会话状态应如何存储到具有会话 ID 的持久且可靠的云存储中，以最小化所需的存储空间。

当调用函数时，用户可以包括会话标识以从保存的状态中恢复。当分配一个新进程时，函数调用可以与它们的会话 ID 一起排队，这样沙盒初始化可以与从存储中下载状态重叠，有助于隐藏有状态函数的启动延迟。

3.2 函数调用处理

PraaS 函数存在于会话的上下文中，正是它们关联的会话决定了它们的处理方式。如果会话处于活动状态，则用户直接连接到正确的通信通道，完全绕过控制平面并确保从用户到函数内存的零拷贝路径。

如果进程没有运行该会话，则涉及控制平面：系统然后检查与该会话匹配的状态信息是否已存在于存储中。如果是，则换入会话，并且调用可以使用会话中存储的所有信息。否则，分配一个新会话。在这两种情况下，允许快速后续调用的通信信道信息被转发给用户。我们的多个函数可以通过多路复用在会话中共享一个通信通道。

本地分派

我们允许用户在每个执行请求中指定对一个或多个先前调用的结果的直接依赖。当指定的依赖项和活动调用之间存在匹配时，执行请求会停止。协同定位调用是 serverless 工作流 [41] 中的一项重要优化技术，依赖项的本地编码避免了与工作流编排器的交互。此外，依赖项允许卸载函数调用，同时保留调用的先进先出顺序。要在当前的 serverless 模型中实现 FIFO 执行，用户必须手动等待上一次调用或使用带有函数触发器的 FIFO 队列。这两种选择都会产生额外的延迟和成本。同时，*PraaS* 函数依赖允许以最小的调用延迟分派函数管道，保持函数代码完整性，并避免不必要的融合。

多源函数

serverless 函数可以被视为一元操作，但实践表明，许多聚合和工作流函数接收来自多个函数前驱的输入。

在 *PraaS* 中，我们为具有可变数量参数的函数提供原生支持。由于会话状态与函数工作者分离，因此状态可以有效地接收和存储来自多个生产者的函数参数。因此，一旦满足所有依赖关系，*PraaS* 就可以启动多源函数。多源函数可用于 serverless 工作流，以有效实现函数之间的数据流。

示例：聚合函数用于在 serverless 工作负载中实现 reduce 和 allreduce 函数 [4, 8, 46]。每个聚合函数接收来自多个前导函数的输入。在 FaaS 中，用户必须使用工作流编排器和云存储触发器来处理聚合，因为不能同时调用函数。FaaS 缺乏对更复杂逻辑的支持，即对存储中的每个新文件调用函数，并且它们必须手动检查是否所有参数都已放入存储中。在 *PraaS* 中，用户定义了一个 n 元 reduce 函数。当客户端通过单个会话调用 reduce 函数时，它们附加相同的调用标识，并且进程控制器对传递的参数进行计数。一旦所有输入参数到达，该函数就会被调用。减少是在单个进程中计算的，从而节省了云存储引入的成本和延迟。

3.3 *PraaS* 数据平面

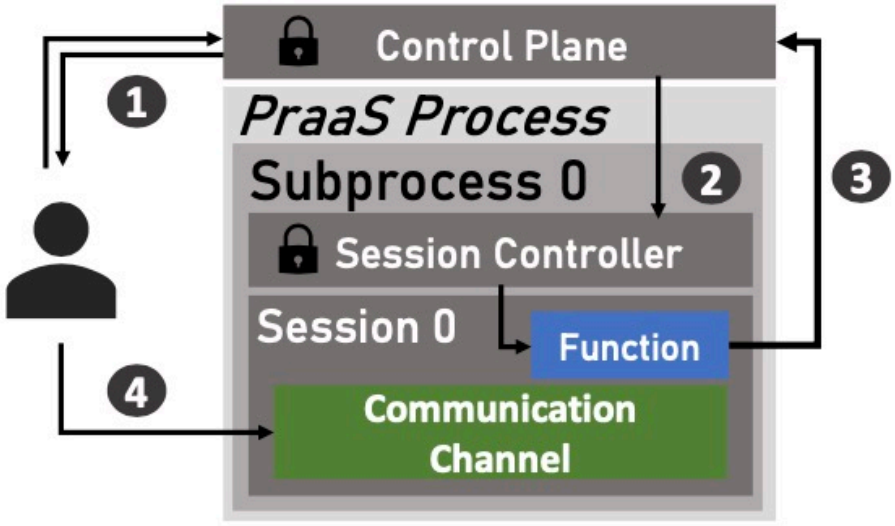


Figure 5: ***PraaS* data plane**: bypassing the control plane allows for zero-copy invocations.

PraaS 通过针对低延迟和高吞吐量调用优化的 serverless 数据平面的新概念帮助最大限度地减少 FaaS 延迟（图 5）。完整的 serverless 调用需要授权请求、选择和可选地分配资源，并将用户的有效负载重定向到函数执行器。当许多执行请求被重定向到同一个暖容器时，重复的控制操作是多余的，然后可以从连续调用中删除。因此，只要授权保持有效，调用就可以完全绕过控制操作，将数据直接从用户移动到进程中的函数执行器。

在 *PraaS* 中，第一次调用函数时（1），控制平面通知会话控制器在可用子进程上分配会话（2），serverless 系统在该点执行所有配置和授权任务。用户接收连接信息以便能够容易且快速地重新连接(3)。只要会话保持活动状态，同一会话中的每个连续函数都会通过与进程 (4) 的直接连接来调用。函数调用从用户接收数据到会话的状态内存，零拷贝方法有助于在更大的负载上实现高吞吐量。因此，调用延迟仅受网络结构和函数线程分配性能的限制。

3.4 伸缩会话

PraaS 进程是一个虚拟的分布式实体。为了确保实用、高效的实现，我们引入了子进程，代表用于托管会话的实际容器。子进程对用户是不可见的，并由云提供商自动管理。如果一个进程中的并发会话数量很高，云提供商可以生成额外的子进程来处理增加的负载，可以删除没有活动会话的子进程，甚至可以将多个子进程的会话压缩到一个子进程以提高资源利用率。为此，该进程还包含一个全局控制器，用于跟踪子进程及其托管的会话。

迁移和压缩（compaction）

跨子进程分布的会话以允许云提供商更好地进行负载平衡和资源检索。与正在迁移的会话具有活动连接的客户端会收到包含迁移详细信息的数据包，它们可以建立与新通信通道的连接。

4 PraaS 中的 serverless 通信

serverless 计算已被用于支持分布式应用程序和工作流，但 FaaS 编程模型的局限性需要在调度和通信方面进行复杂且特定领域的优化 [47-49]。serverless 计算的大多数进步都集中在优化南北通信而不是东西通信。

FaaS 应用程序包括一些通信模式 - 消息传递，通常实现为将函数的输出传播到连续调用的输入 [49]； 来自许多函数的输出的聚合 [46, 50]，并使用缓存在附近实例中的数据来绕过存储和更快的访问 [42, 51]。

相比之下，*PraaS* 提供全局内存，一种高效且直接的 serverless 应用程序通信模型（第 4.1 节），绕过云存储但由云存储提供支持。我们的通信模型不专注于进程调用，因为它们的生命周期有限，但它专注于数据移动操作，允许动态可重构的应用程序从点对点通信中受益。

4.1 全局内存

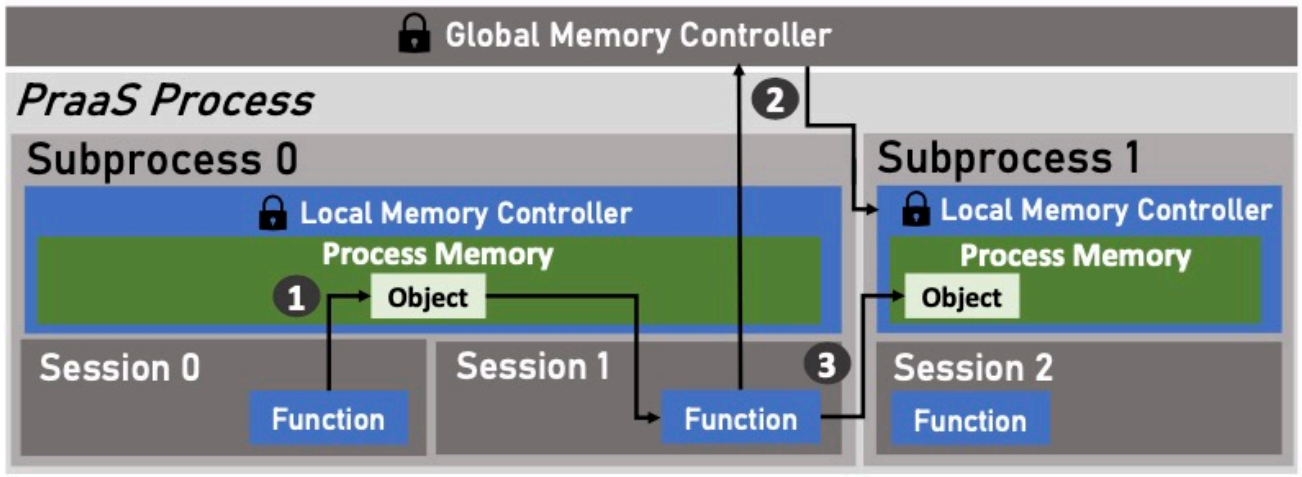


Figure 6: *PraaS* Global Memory.

在 *PraaS* 中，我们提供了一个全局通信抽象层，以实现优化的进程会话间数据交换并隐藏 serverless 通信的复杂性。该通信层受到 MPI 远程内存访问 [52] 的启发。我们在图 6 中展示了全局内存系统的概述。除了可以私有使用的内存会话（保证位于会话所在的子进程上）之外，会话还可以写入全局内存。系统不保证分配全局内存区域的子进程。全局控制器存储全局内存对象的位置。全局内存由持久云存储备份。内存对象通过内存接口访问，该接口隐藏了通过存储和点对点通信进行的数据操作。

全局控制器

全局内存布局通过用直接通信替换存储操作来实现 serverless 中的消息传递范式。我们包括一个全局控制器服务作为系统的外部持久组件（图 6），并使其负责跨子进程监视全局内存对象位置，并将会话中的请求中继到子进程内的本地内存控制器。

接口

进程函数使用类似于 OS 级进程中的内存寻址操作的简单接口与全局进程内存交互。特别是，读写操作接受来自全局命名空间的标识符（想想操作系统级进程中的地址空间）。也可以使用类似于 malloc 和 free 的接口来分配和取消分配全局内存。

通信

基于存储的通信模式缓慢且昂贵 [10,53,54]，但由于 worker 的短暂性，FaaS 中的函数间通信难以维护。

会话可以通过读取和写入全局进程内存中的内存对象（1）与其他会话进行通信。为了建立这样的通信信道，会话向全局通信控制器(2)请求对象的位置。全局通信控制器指示托管对象的子进程和发起会话的本地内存控制器创建通信通道（3）。因此，任何此类操作都不需要发送方和接收方分别将更新推送到外部服务并轮询更改。由于全局内存对象识别通信目标，我们建立消息传递而不识别和枚举临时和不可靠的函数。

此外，由于 p2p 通信实现了更高的带宽和更低的成本，因此通信用进程会话之间的共享对象代替了存储读取操作。

例子：我们考虑一组计算部分结果的函数，应用全局归约来同步进度，并继续下一批任务，就像分布式机器学习训练中的情况一样 [4]。在 FaaS 中，函数必须借助存储来传达结果。此外，缺少状态意味着函数被再次调用以进行迭代并且用户接受重新初始化的惩罚，或者人为地保持函数处于活动状态以保持数据局部性并且它们在通信轮次中因不必要的等待而产生成本。*PraaS* 模型提供了一种更实惠且同时更高效的替代方案：函数将与其部分结果对应的对象存储在共享会话或进程内存中，允许它们将归约数据直接移动到目的地，并且它们停止产生计算之后立即收费。然而，由于会话状态不会立即被驱逐，函数会受益于温暖的环境和下一次迭代中更好的数据局部性。

动态子进程成员资格

在 *PraaS* 中，云提供商可以触发对子进程配置的更改，以根据需求扩大或缩小系统。每个更改都必须是原子的，以确保交换消息的有效性。全局内存控制器存储全局内存对象在所有子进程中的位置，并且当整个子进程配置发生变化时会更新此信息。我们将其定义为一个时代变化，并导致两步操作。控制器向所有子进程宣布配置更改是必要的，子进程通知它们的会话。一旦处理了最后一个时期的所有未完成的通信，会话就会通知它们的子进程。然后，子进程通知控制器，更改配置，然后通知子进程及其会话所有子进程会话的位置。因此，状态的每一次变化都开始一个新的纪元，全局内存控制器将纪元变化公告分发给所有参与的子进程。更改通知允许应用程序优雅地重新调整消息传递策略并避免在临时 worker 存在的情况下发生数据故障。

5 *PraaS* 实践

在本节中，我们将讨论如何在实践中实现我们提出的设计，重点关注四个主要方面：为我们的运行时服务的容器、控制平面、全局内存如何工作以及客户端库。

5.1 控制平面

控制平面的主要职责是接受用户请求并管理进程和会话。系统的外向部分是具有以下函数的 HTTP 前端：

- **分配会话**创建一个新会话。分配必须指定会话可以使用的最大内存量。会话托管在具有足够资源支持它的子进程上；
- **检索会话**从存储中加载会话状态；
- **invoke** 分配一个新会话或检索一个现有会话并传递函数调用有效负载；
- **进程管理支持**，例如创建新进程或删除进程。

进程、会话和调用都是唯一标识的，并且具有层次结构：调用托管在会话上，会话托管在子进程上，一组子进程形成一个进程。

控制平面依赖于内存中的 Redis 缓存来存储有关哪些会话属于哪个进程以及哪些会话是活动的以及哪些会话被交换的信息。

执行后端

子进程由系统管理，并随着 *PraaS* 系统上的负载变化而按比例放大和缩小。为了运行和管理运行这些子进程的容器，我们部署了连接到控制平面的执行服务器。如果系统负载高，执行器服务器会满足创建额外子进程的请求，或者满足来自多个低负载子进程的压缩会话的请求。可以以循环方式将子进程分配给服务器，但也可以实施其他优化的分配策略。我们的系统在这方面没有任何限制，因此可以支持像 Lambda 中的低延迟调度程序 [29]。执行器服务器可以采用 VM 或 serverless 函数的形式。

子进程

子进程运行一个会话控制器，它接受来自控制平面的请求并管理本地会话。当收到分配请求时，它会分叉一个新的会话。它还向控制平面报告会话被换出或删除。子进程还运行本地内存控制器，我们将在 5.3 节中详细介绍。

5.2 会话

会话的主要组件是可以交换到云存储（我们的实现中为 AWS S3）的本地可用内存，以及数据平面 TCP 连接。调用负载直接接收到内存中，并且可以被调用的函数访问。

为了运行用户代码，线程池分派调用的函数。每个会话都有一个可以由云提供商设置的活动函数数量的上限，例如与使用的内存有关 - 将线程池限制为 N 个线程。超出此数量排队的函数将等待资源可用。

会话通过会话内存为会话控制器提供使用指标。控制器使用这些数据平面调用指标来决定何时不再使用会话并且可以换出。当用户关闭与她的连接时，会话也会关闭。

代码部署

我们建议用户使用在 *PraaS* 系统中部署所需的代码和库来定义容器。此容器由我们的监控系统扩展，并以受限权限运行，类似于 Lambda C++ 容器。只有 *PraaS* 控制器以 root 权限运行。

5.3 全局内存

为了实现跨子进程物理分布的内存的全局视图，每个子进程运行一个本地内存控制器，该控制器充当一个简单的服务器来响应会话对对象的创建、放置、获取和删除操作的请求。虽然接口很简单，但背后的机制更复杂：当会话请求一个对象时，它会首先从本地内存控制器请求它。如果对象不在同一个子进程上，则向全局内存控制器发出请求，并开始 NAT 打孔操作。全局内存控制器运行一个页表，其中包含将对象链接到由 {Object name, Location} 等元组组成的子进程的信息。全局内存通知拥有该对象的本地内存控制器在其一侧使用 NAT 打孔来完成连接。一旦连接存在，未来的通信就可以在不涉及全局内存控制器的情况下进行。

5.4 计费和账户

与当前的 FaaS 和 IaaS 平台类似，*PraaS* 允许用户使用预定义的内存和 CPU 资源创建会话。除了指定本地会话的资源外，用户还需要指出进程内存应该有多大。为了使这项任务更简单，*PraaS* 允许用户指定每个会话的进程内存增量，以便总进程内存随会话数线性增长。将会话内存与进程内存分配分开可以提高资源利用率并减少为未使用资源支付的金额 [11]。最后，为了支持会话交换，*PraaS* 还向用户收取保存交换会话所需的存储费用。

调用记帐由子进程执行，这些子进程在其生命周期内收集用于计算、内存使用和 I/O 的使用指标。为此，我们在子进程上使用 Docker 容器指标，并在所有子进程中对它们进行整理，以实现远程可观察性和精细计费。

5.5 客户端库

客户端库实现 HTTP 请求的用户端来管理进程和会话，二进制数据作为有效负载通过 TCP 连接传输。客户可以使用他们选择的过程来序列化数据。客户端库的另一个特性是，如果操作超时或对象或会话不可用，用户可以实现要执行的处理程序。

5.6 实现 *PraaS*

我们将 *PraaS* 实施为现有 CaaS 和 FaaS 平台之上的一组扩展，以促进在现有云系统中的广泛采用。云提供商自动管理运行专用 *PraaS* 工作者的进程容器队列（第 5.1 节）。特别是，我们在 Knative [55] 上演示了 *PraaS* 的原型实现，Knative [55] 是一个使用 Kubernetes [56] 部署的 serverless 平台（第 5.6 节）。但是，我们修改容器和调度策略的设计也适用于其他平台，并且可以无缝移植到其他系统。

对现有平台的更改涉及多个方面。必须调整调度以使用内存中的缓存来检查进程和会话 ID 以做出决策。子进程扩展决策必须采用会话提供的数据平面指标，而不是消除具有活动会话的子进程。k8s 支持通过删除成本机制删除选定的容器进行缩减。最后，可以部署我们的控制平面来提供会话进程API，并在需要时使用kubernetes API分配容器。

6 Evaluation

在本节中，我们评估 *PraaS*。在我们的实验中，我们总是重复每个测量至少 100 次，并且绘制中位数以及 99% 的非参数 CI。在评估的每个部分，我们都专注于回答有关 *PraaS* 的相关问题。

6.1 数据平面

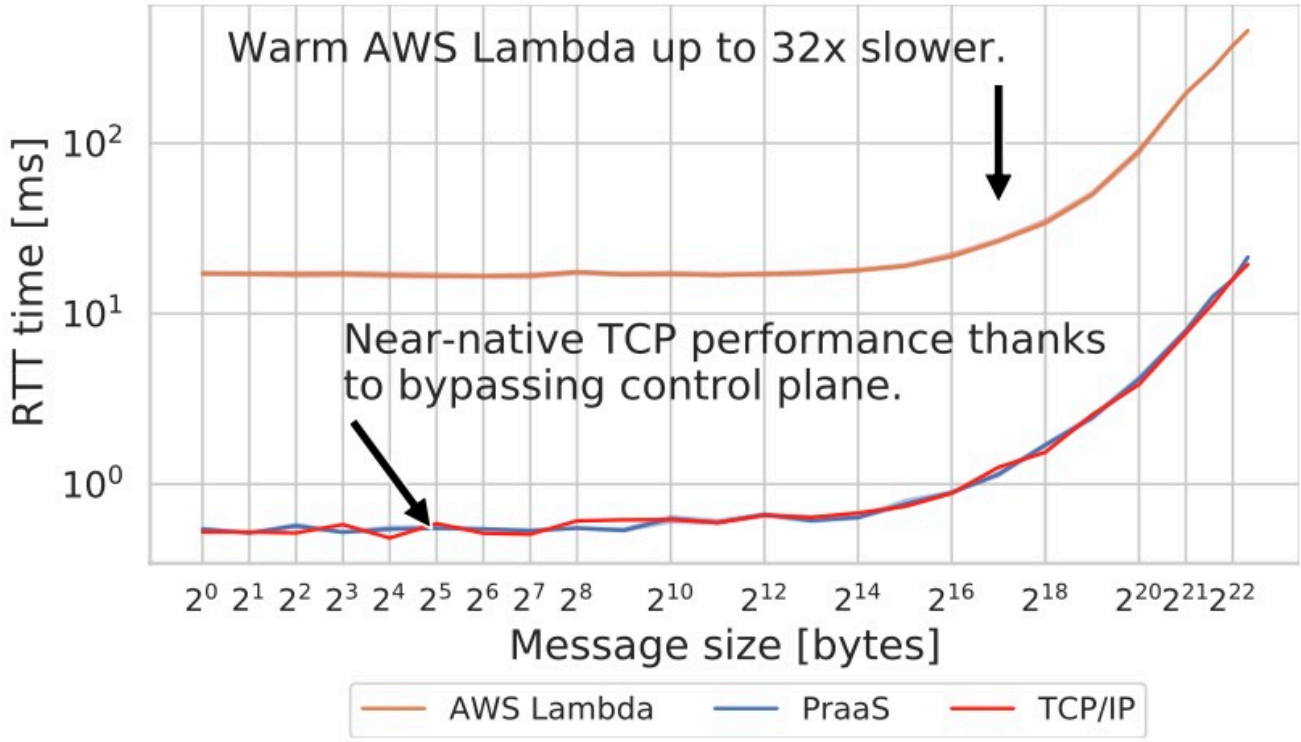


Figure 7: PraaS vs FaaS: bypassing control plane.

数据平面是否改善了调用延迟?

我们的方法通过使用优化的 serverless 数据平面允许快速直接调用。为了评估这一说法，我们比较了在 *PraaS* 中处理具有各种有效负载大小的调用的往返时间与在 AWS Lambda 上处理相同调用的往返时间。基准测试从 t3.medium VM 执行，并调用温暖的 AWS Lambda 和 *PraaS* 函数。图 7 中的结果显示了使用 *PraaS* 的一致且显着的优势，因为 AWS Lambda 大约慢了 32 倍。AWS Lambda 开销随着负载大小的增加而显着增加，这表明关键路径上有多个负载副本，这使得大负载调用变得越来越慢。

此外，我们通过提供使用 netperf 收集的 TCP 基线来在用户和执行服务器之间传递有效负载本身，从而证明 *PraaS* 有效地没有开销。因此，**绕过控制平面**可以显着改善函数调用的往返时间。

6.2 快速函数通信

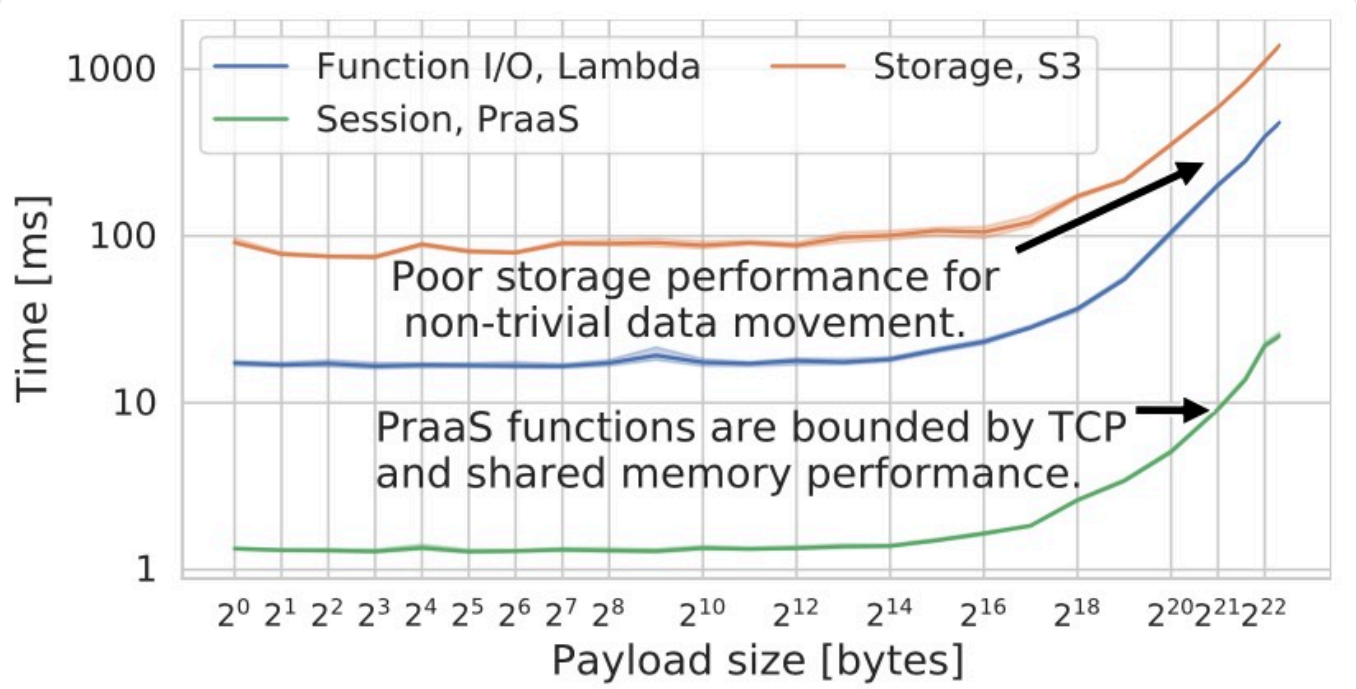


Figure 8: PraaS vs FaaS: comm. between functions.

使用会话内存进行函数之间的通信是否会改善延迟?

serverless 工作流中的一个重要概念是链接函数以将一个输出作为输入传递给另一个。我们声称使用会话内存是允许函数进行通信的一种快速方法。我们通过启动一个具有不同输入大小的函数来评估这一点，将结果传递给另一个函数，该函数又将数据返回给用户。为此，Dynamo DB 价格昂贵且过于有限（最大元素大小为 400 kB），因此不能被视为一种选择。剩下的选项是诉诸访问 S3 存储或使用 Lambda 的函数 I/O。图 8 显示，使用 *PraaS* 会话内存进行函数通信比使用 S3 存储处理任何数据大小要快两个数量级，并且比 Lambda 函数 I/O 快大约十倍。结果并不令人惊讶，并强调需要快速的本地通信，而不是依赖云存储解决方案进行函数通信。使用会话内存作为媒介的函数之间的快速通信比现有的替代方案快得多。

6.3 会话管理

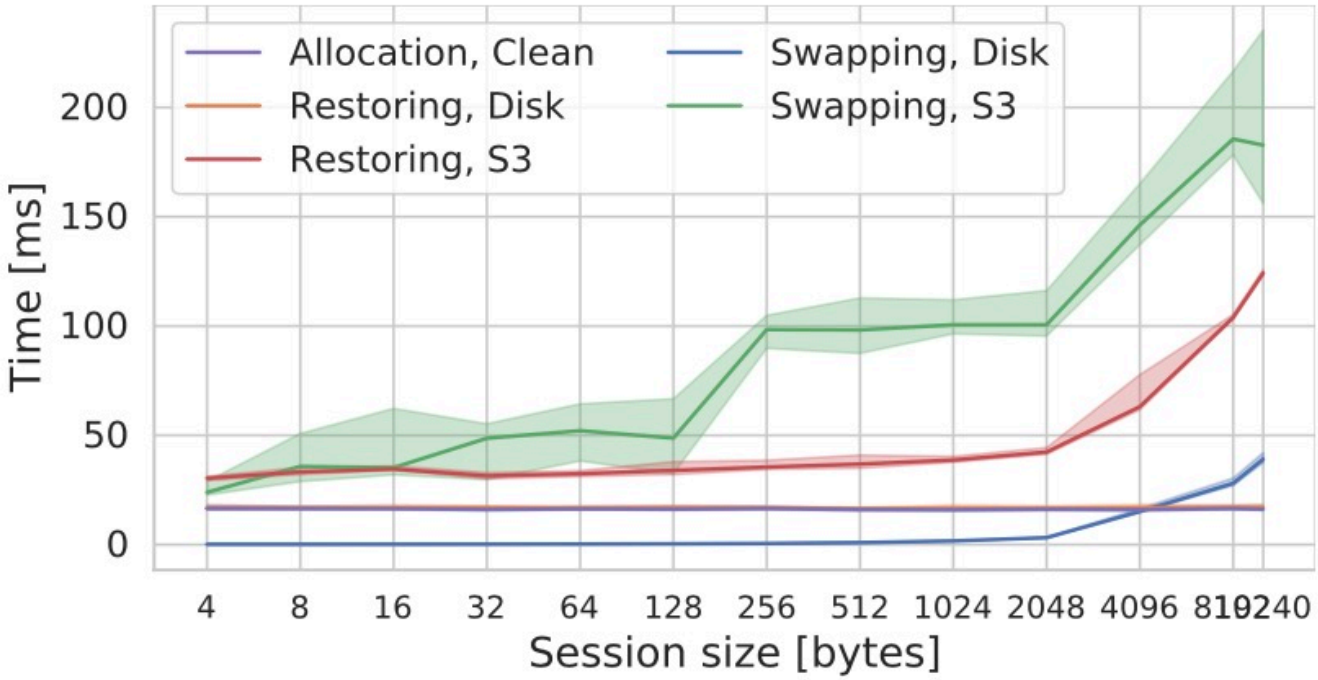


Figure 9: Overhead of session allocation and swapping.

交换会话是否会产生高额开销?

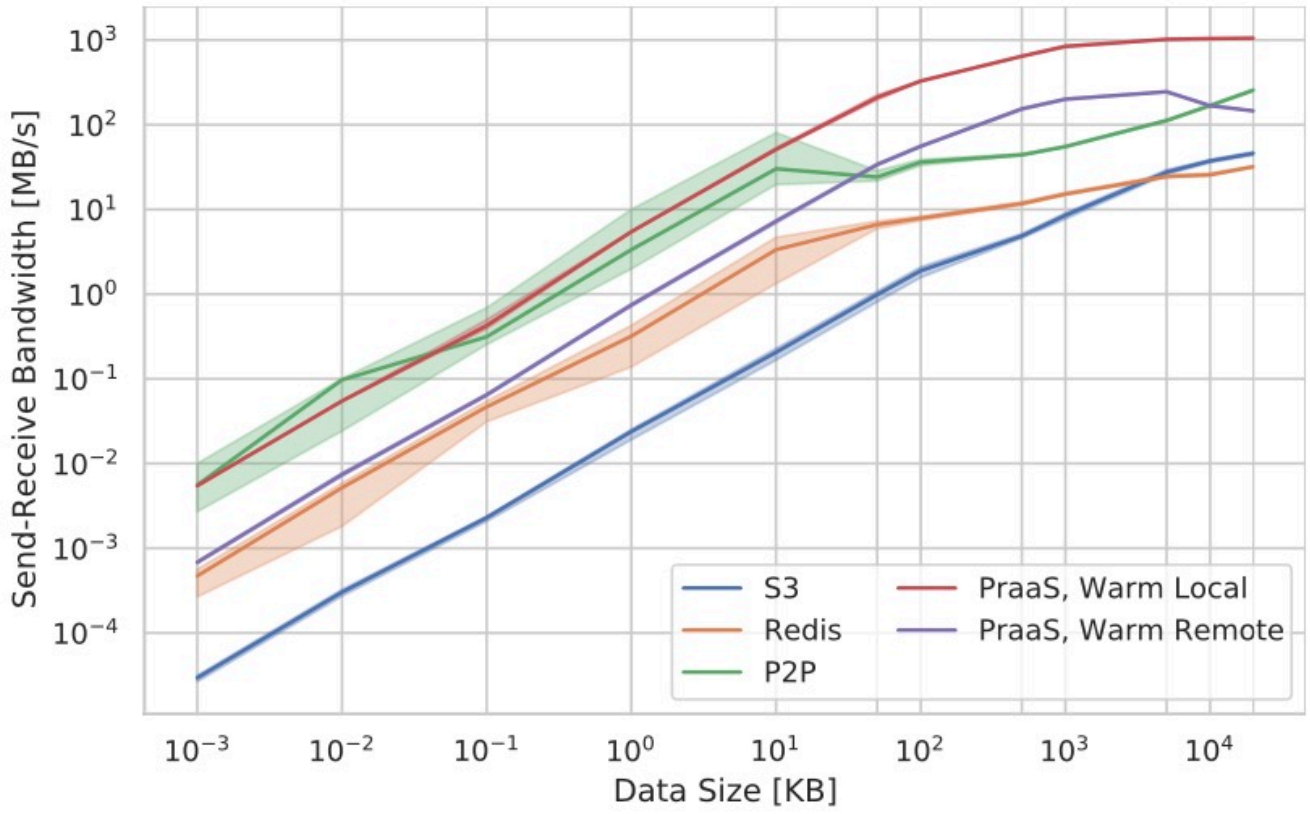
为了实际有用，我们用于交换会话进出存储的系统必须是高效的。为了评估它，我们测量了分配一个干净的会话、恢复一个交换的会话或换出一个正在运行的会话所需的时间。

我们将会话交换到 S3 存储或连接到 EC2（通用 SSD）的 Elastic Block Storage 的 gp2 实例。在后一个版本中，云提供商可以将会话交换到连接到虚拟机的磁盘上，以消除从关键路径交换的开销，并将交换的会话异步存储到 S3 存储。S3 具有非常高的耐用性，EBS 的年故障率为 0.1-0.2% [57]。

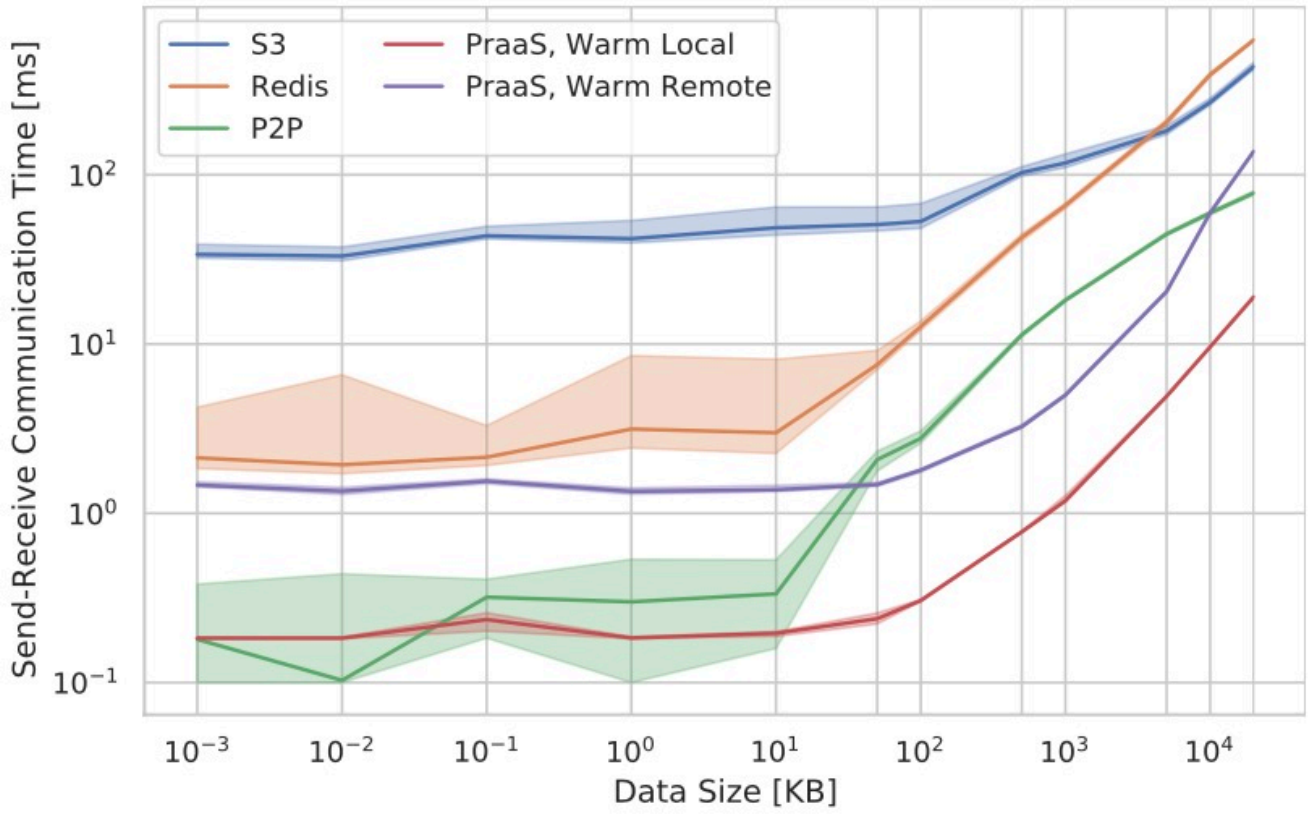
这具有进一步的好处，即从磁盘恢复时可以快速恢复会话。恢复会话的延迟比分配容器的延迟低 [58, 59] 并且可以同时完成，有效地隐藏了从内存中加载状态所需的时间。

如图 9 所示，从磁盘恢复会话与分配新会话所需的时间相同，即使从 S2 恢复也只需不到 150 毫秒，即使对于最大的会话大小也是如此。换出一个会话也只需要 25 到 200 毫秒，具体取决于会话大小。因此，会话交换**不会引入显着的开销**。

6.4 全局内存



(a) Bandwidth comparison



(b) Latency comparison

Figure 10: Latency and bandwidth for communication

between Lambda functions and the global memory put-get operations in *PraaS*.

与使用 S3 存储相比，通过全局进程内存进行通信是否会提高性能?

我们通过将其性能与通过 S3、Redis 甚至建立直接通信通道的 Lambda 函数通信进行比较来评估使用全局进程内存的会话之间的通信。所有测量均在具有 2 GB 内存的 Lambda 实例上执行。对于 *PraaS*，我们测量执行相同 get 和 put 操作的函数。我们区分暖本地场景 - 数据在本地内存控制器和暖远程 - 数据在不同机器上的另一个子进程上。对另一台机器的平均 ping 时间为 350 微秒。

带宽（图 10a）和延迟（图 10b）的结果显示了使用 *PraaS* 的明显优势。显示类似结果的唯一解决方案是使用 Lambda 函数的直接 p2p 连接。我们预计它会比 S3 和 Redis 更快，因为它不涉及额外的服务，只需一份数据副本。但是，这种方案不适合 serverless 范式——函数在传输时必须在线，并且它们必须知道彼此的存在，使得实际使用变得困难。此外，我们确认了先前的结果，即 Lambda 显示出显着的 I/O 可变性 [11, 35]。*PraaS* 甚至可以跨物理分离的机器提供**快速、高效的通信**。

6.5 开销

	V_m	C_m	F_m	G_V	G_C
AWS	$3.53 \cdot 10^{-3}$	$4.45 \cdot 10^{-3}$	$1.5 \cdot 10^{-2}$	76.43%	70.37%
Azure	$3.87 \cdot 10^{-3} \pm 0.005$	$4.45 \cdot 10^{-3}$	$1.14 \cdot 10^{-2}$	$66.05\% \pm 0.05$	60.96%

Table 3: Decreased costs of *PraaS* sessions in comparison to FaaS provisioned instances.

系统开销是哪些?

PraaS 用户必须为保持会话状态而付费，并且成本应该线性地取决于活动会话的生命周期。现代商业 FaaS 系统上唯一可比较的函数是预置函数实例，称为 AWS Lambda 上的预置并发和 Azure Functions 的高级计划。云提供商保证准备好函数实例以降低冷启动频率。虽然可以说这些函数不是 serverless 的，但这些实例可以被视为温暖和低延迟状态 1 的有限替代品。除了为使用计算资源付费外，用户还需支付取决于内存大小和资源供应持续时间的活动状态费用 Fs。

为了估计将 *PraaS* 会话保持在内存中的成本，我们使用其他云服务的内存作为云运营商成本的代理。首先，我们比较了从计算优化到内存优化的虚拟机实例的成本 Vm，并将其除以获得的内存大小，从而估算获得 1 GB DRAM 的额外成本。我们比较了 AWS 上的 c6g 和 x2gd 实例以及 Azure 上的 Fs 和 Eadsv5 系列，我们发现每个实例大小的 Vm 几乎相同。然后，我们选择在托管容器系统 Cm 上部署 *PraaS* 时分配额外内存的成本，并考虑 AWS Fargate 和 Azure 容器实例。通过将 *PraaS* 部署的内存成本 Vm、Cm 与 FaaS 上预置存储 Fm 的成本进行比较，我们分别估计

了将会话状态从预置 FaaS 移动到 VM 和容器的成本降低 GV 和 GC。表 3 中的结果证明，可以以较低的成本提供 *PraaS* 会话，最多可降低 76%，并且估算涵盖了包含在 VM 实例价格中的云提供商成本和利润。此外，内存优化实例附带额外的 SSD 存储，可用于为交换会话实施低延迟层，无需额外成本。

6.6 函数

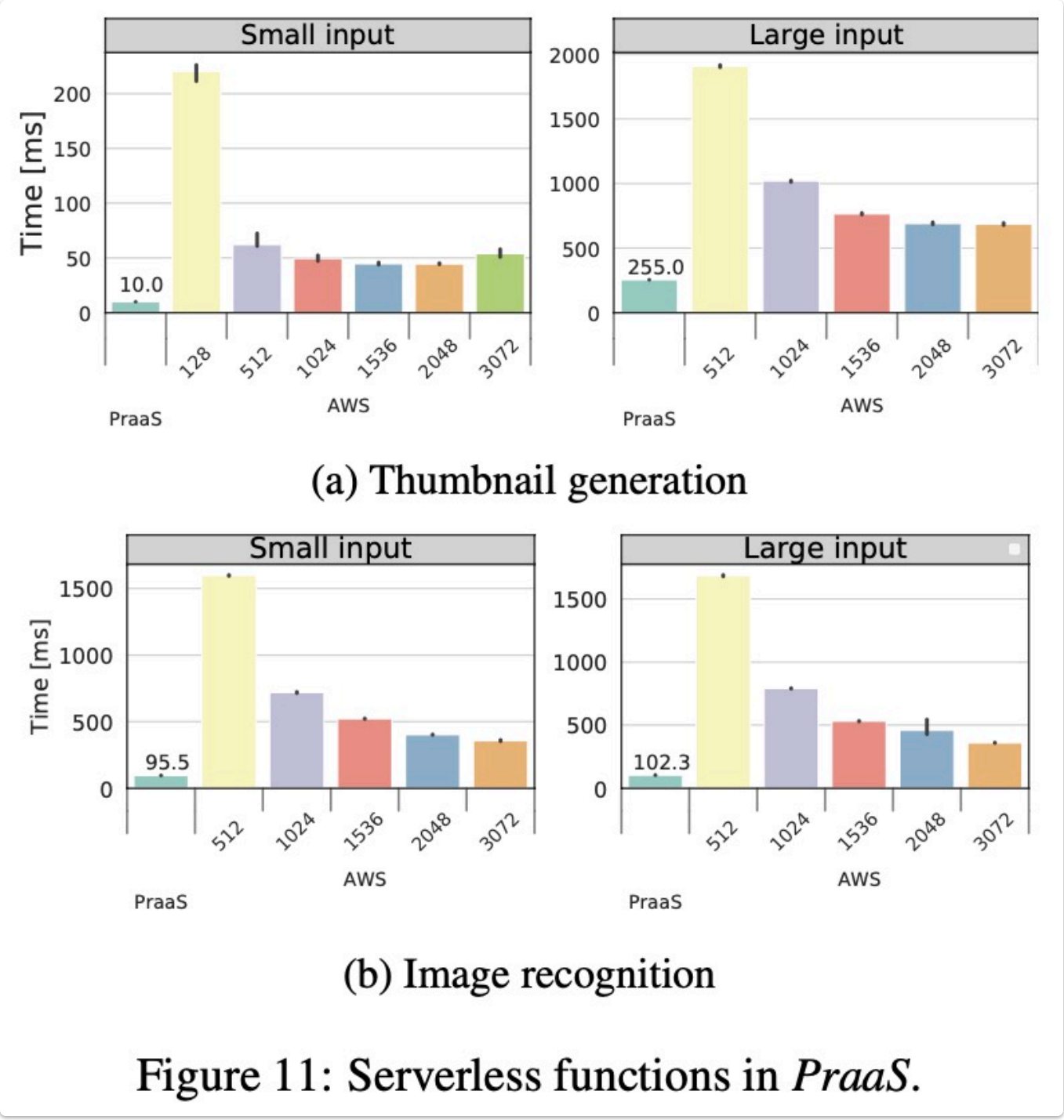


Figure 11: Serverless functions in *PraaS*.

FaaS 函数可以在 *PraaS* 上运行吗?

我们选择缩略图生成器作为通用图像处理的示例，并选择执行 ResNet-50 预测的图像识别基准作为将深度学习推理集成到应用程序中的示例，两者均来自 SeBS 基准 [11]。

我们使用 OpenCV 4.5 实现缩略图生成，并使用两个图像评估函数，一个 97 kB 和一个 3.6 MB。对于 AWS Lambda 函数，我们需要在 POST 请求中将二进制图像数据作为 base64 编码的字符串提交，这会由于编码和转换而增加大量开销。另一方面，*PraaS* 函数受益于不受云 API 要求约束的有效负载格式。

图像识别是在 PyTorch C++ API 的帮助下使用 OpenCV 4.5、libtorch 1.9 和 torchvision 0.1 实现的。我们使用两个输入来评估函数，一个 53 kB 的图像和一个 230 kB 的图像。结果（图 11a 和 11b）证明 *PraaS* 优于我们比较的所有 AWS 实例。

6.7 讨论

鉴于 *PraaS* 提供的函数，它可以用作一个平台，以异步和同步方式实现具有梯度平均随机梯度下降的分布式逻辑回归。由于使用直接通信，与当前的实现[60]相比，系统的性能可以提高。*PraaS* 模型应该更容易表达异步计算。最后，*PraaS* 可以动态地改变在给定时间被调用的函数的数量，而 LambdaML [60] 在整个执行过程中依赖于固定数量的函数。

7 相关工作

有状态的函数通过允许函数保持状态（即使是分离的）来打开可以从 FaaS 中受益的应用程序的范围。研究人员已经在专门用于 serverless [18、61] 的键值存储和弹性临时缓存 [51、62、63] 之上构建了状态函数，这些缓存结合了不同的放置策略来管理成本和性能。然而，对于许多应用程序来说，有状态是不够的，因为函数可能随时失败。因此，已经提出了诸如 Beldi [19] 和 Boki [20] 之类的系统，以帮助开发人员在临时函数之上构建一致且容错的系统。*PraaS* 没有提供另一种存储选项来保持函数状态或提出一种新技术来处理故障，而是通过提出保留状态的会话的概念在 IaaS 和 FaaS 之间取得新的平衡。在一个进程中，会话可以交换消息。*PraaS* 通过将状态划分为由会话管理的所有权域，消除了有状态函数中状态同步的需要。会话可以随时交换，但它们的状态不会丢失。

函数编排和数据局部性也正在被广泛研究。Speedo [64] 和 Nightcore [65] 等系统通过加速控制平面 [64] 或完全跳过控制平面 [65] 进行内部函数调用来优化函数编排。其他系统已经研究了如何通过比较不同的函数通信策略和自动调整部署决策来优化数据路径[66]，或者通过允许多个函数随着时间的推移共享执行环境来避免移动数据[67]。

另一方面，*PraaS* 建议用户应该能够直接连接到会话，从而在第一个初始化步骤之后完全避免控制平面。此外，通过在会话之间划分状态，进程函数可以直接在它们之间交换数据，而不是依赖先前系统提出的外部存储[7, 8]。尽管尽可能利用函数之间的直接通信，但 *PraaS* 为用户提供了全局内存空间抽象，而不是直接向开发人员公开网络原语 [68]。

serverless 沙箱与传统的虚拟机管理器相比，利用专门的虚拟化引擎 [58, 69] 大大减少了启动时间和内存占用。然而，为了继续提高 serverless 应用程序的可扩展性和弹性，已提出软隔离系统 [70-72] 在同一操作系统进程内共同执行多个调用。*PraaS* 部分地受到此类系统的启发，允许多个函数在单个会话中同时执行。这样做可以减少资源冗余，并出现新的本地通信机会。然而，

PraaS 的设计并不排除其他正交优化技术，例如用于优化会话启动时间和内存占用的图像预初始化 [69、73-75]，或用于优化进程启动和内存占用的 unikernels [76-78]。

8 结论

这项工作提出了 *PraaS*，这是一种新的云计算抽象，可为临时 worker 带来持久状态。通过利用进程和会话，应用程序现在可以受益于低延迟状态、绕过控制平面的快速调用以及会话之间的快速通信。我们展示了 *PraaS* 是一种有效的抽象，可以通过直接连接和快速通信选项来实现有状态函数。

引用

[1] Yanqi Zhang, ´Inigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and cheaper serverless computing on harvested resources. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21, page 724–739, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387095. doi: 10.1145/3477132.3483580. URL https://doi.org/10.1145/3477132.3483580.

[2] Amoghavarsha Suresh and Anshul Gandhi. Servermore: Opportunistic execution of serverless functions in the cloud. SoCC '21, page 570–584, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450386388. doi: 10.1145/3472883.3486979. URL https://doi.org/10.1145/3472883.3486979.

[3] Marcin Copik, Konstantin Taranov, Alexandru Calotoiu, and Torsten Hoefler. RFaaS: RDMA-Enabled FaaS Platform for Serverless High-Performance Computing, 2021. URL https://arxiv.org/abs/2106.13859.

[4] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. Towards demystifying serverless machine learning training. In ACM SIGMOD International Conference on Management of Data (SIGMOD 2021), June 2021.

[5] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. Sprocket: A serverless video processing framework. In Proceedings of the ACM Symposium on Cloud Computing, SoCC '18, pages 263–274, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360111. doi: 10.1145/3267809.3267815. URL https://doi.org/10.1145/3267809.3267815.

[6] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17),

pages 363–376, Boston, MA, March 2017. USENIX Association. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi.

[7] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. Starling: A scalable query engine on cloud functions. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, page 131–141, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3380609. URL https://doi.org/10.1145/3318464.3380609.

[8] Ingo Muller, Renato Marroquin, and Gustavo Alonso. Lambada: Interactive data analytics on cold data using serverless cloud infrastructure. ArXiv, abs/1912.00937, 2019.

[9] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pages 193–206, Boston, MA, February 2019. USENIX Association. ISBN 978-1-931971-49-2. URL https://www.usenix.org/conference/nsdi19/presentation/pu.

[10] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. Cloud programming simplified: A berkeley view on serverless computing. CoRR, abs/1902.03383, 2019. URL http://arxiv.org/abs/1902.03383.

[11] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. Sebs: A serverless benchmark suite for function-as-a-service computing. In Proceedings of the 22nd International Middleware Conference, Middleware '21. Association for Computing Machinery, 2021. doi: 10.1145/3464298.3476133. URL https://doi.org/10.1145/3464298.3476133.

[12] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. What serverless computing is and should become: The next phase of cloud computing. Commun. ACM, 64(5):76–84, April 2021. ISSN 0001-0782. doi: 10.1145/3406011. URL https://doi.org/10.1145/3406011.

[13] L. Feng, P. Kudva, D. Da Silva, and J. Hu. Exploring serverless computing for neural network training. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pages 334–341, July 2018. doi: 10.1109/CLOUD.2018.00049.

[14] Hao Wang, Di Niu, and Baochun Li. Distributed machine learning with a serverless architecture. In IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pages 1288–1296. IEEE, 2019.

[15] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, et al. Dorylus: Affordable, scalable, and accurate {GNN} training with distributed {CPU} servers and serverless threads. In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), pages 495–514, 2021.

[16] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. Cirrus: A serverless framework for end-to-end ml workflows. In Proceedings of the ACM Symposium on Cloud Computing, pages 13–24, 2019.

[17] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'18, pages 427–444, USA, 2018. USENIX Association. ISBN 9781931971478.

[18] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph E. Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. Cloudburst: Stateful functions-as-a-service. Proc. VLDB Endow., 13(12):2438–2452, July 2020. ISSN 2150-8097. doi: 10.14778/3407790.3407836. URL https://doi.org/10.14778/3407790.3407836.

[19] Haoran Zhang, Adney Cardoza, Peter Baile Chen, Sebastian Angel, and Vincent Liu. Fault-tolerant and transactional stateful serverless workflows. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pages 1187–1204. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/zhang-haoran.

[20] Zhipeng Jia and Emmett Witchel. Boki: Stateful serverless computing with shared logs. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21, page 691–707, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387095. doi: 10.1145/3477132.3483541. URL https://doi.org/10.1145/3477132.3483541.

[21] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In 2020 USENIX Annual Technical Conference (USENIX ATC 20), pages 205–218. USENIX Association, July 2020. ISBN 978-1-939133-14-4. URL https://www.usenix.org/conference/atc20/presentation/shahrad.

[22] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pages

1–16, Broomfield, CO, October 2014. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/conference/osdi14/technical-sessions/presentation/peter.

[23] Collin Lee and John Ousterhout. Granular computing. In Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS '19, page 149–154, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367271. doi: 10.1145/3317550.3321447. URL https://doi.org/10.1145/3317550.3321447.

[24] Pedro Garcia Lopez, Aleksander Slominski, Michael Behrendt, and Bernard Metzler. Serverless predictions: 2021-2030, 2021.

[25] AWS Lambda. https://aws.amazon.com/lambda/, 2014. Accessed: 2020-01-20.

[26] Azure Functions. https://azure.microsoft.com/en-us/services/functions/, 2016. Accessed: 2020-01-20.

[27] Google Cloud Functions. https://cloud.google.com/functions/, 2017. Accessed: 2020-01-20.

[28] IBM Cloud Functions. https://cloud.ibm.com/functions/, 2016. Accessed: 2020-01-20.

[29] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), pages 419–434, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/agache.

[30] Johannes Manner, Martin EndreB, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), pages 181–188, 2018.

[31] Anup Mohan, Harshad Sane, Kshitij Doshi, Saikrishna Edupuganti, Naren Nayak, and Vadim Sukhomlinov. Agile cold starts for scalable serverless. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), Renton, WA, July 2019. USENIX Association. URL https://www.usenix.org/conference/hotcloud19/presentation/mohan.

[32] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In Proceedings of the 21st International Middleware Conference, Middleware '20, page 356–370, NewYork, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381536. doi: 10.1145/3423211.3425690. URL https://doi.org/10.1145/3423211.3425690.

[33] Paulo Silva, Daniel Fireman, and Thiago Emmanuel Pereira. Prebaking functions to warm the serverless cold start. In Proceedings of the 21st International Middleware Conference, Middleware '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

ISBN 9781450381536. doi: 10.1145/3423211.3425682. URL https://doi.org/10.1145/3423211.3425682.

[34] Kun Suo, Junggab Son, Dazhao Cheng, Wei Chen, and Sabur Baidya. Tackling cold start of serverless applications by efficient and adaptive container runtime reusing. In 2021 IEEE International Conference on Cluster Computing (CLUSTER), pages 433–443, 2021. doi: 10.1109/Cluster48925.2021.00018.

[35] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the curtains of serverless platforms. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '18, page 133–145, USA, 2018. USENIX Association. ISBN 9781931971447.

[36] Adam Eivy and Joe Weinman. Be wary of the economics of "serverless" cloud computing. IEEE Cloud Computing, 4(2):6– 12, 2017. doi: 10.1109/MCC.2017.32.

[37] Apache OpenWhisk. https://openwhisk.apache.org/, 2016. Accessed: 2020-01-20.

[38] M. Sciabarra.` Learning Apache OpenWhisk: Developing Open Serverless Solutions. O'Reilly Media, Incorporated, 2019. ISBN 9781492046165.

[39] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. Wukong: A scalable and localityenhanced framework for serverless parallel computing. In Proceedings of the 11th ACM Symposium on Cloud Computing, SoCC '20, page 1–15, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381376. doi: 10.1145/3419111.3421286. URL https://doi.org/10.1145/3419111.3421286.

[40] Arjun Singhvi, Kevin Houck, Arjun Balasubramanian, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. Archipelago: A scalable low-latency serverless platform, 2019.

[41] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. Sand: Towards high-performance serverless computing. In Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '18, pages 923–935, USA, 2018. USENIX Association. ISBN 9781931971447.

[42] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. Restructuring serverless computing with data-centric function orchestration. arXiv preprint arXiv:2109.13492, 2021.

[43] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '05, page 13, USA, April 2005. USENIX Association.

[44] J. L. Eppinger. TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem. Carnegie Mellon University, Technical Report, ISRI-05-104, January 2005.

[45] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD), pages 199–208, 2019. doi: 10.1109/CLOUD.2019.00043.

[46] V. Gimenez-Alventosa, Germ ´ an Molt ´ o, and Miguel Caballer. A ´ framework and a performance assessment for serverless mapreduce on aws lambda. Future Generation Computer Systems, 97: 259–274, 2019. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2019.02.057. URL https://www.sciencedirect.com/science/article/pii/S0167739X18325172.

[47] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–15, 2020. doi: 10.1109/SC41405.2020.00073.

[48] Minchen Yu, Zhifeng Jiang, Hok Chun Ng, Wei Wang, Ruichuan Chen, and Bo Li. Gillis: Serving large neural networks in serverless functions with automatic model partitioning. In 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), pages 138–148, 2021. doi: 10.1109/ICDCS51616.2021.00022.

[49] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. {SONIC}: Application-aware data passing for chained serverless applications. In 2021 {USENIX} Annual Technical Conference ({USENIX}{ATC} 21), pages 285–301, 2021.

[50] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, NSDI'17, pages 363–376, USA, 2017. USENIX Association. ISBN 9781931971379.

[51] Francisco Romero, Gohar Irfan Chaudhry, ´Inigo Goiri, Pragna Gopa, Paul Batum, Neeraja J. Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, and Ricardo Bianchini. Faa$t: A transparent auto-scaling cache for serverless applications, 2021.

[52] Torsten Hoefler, J. Dinan, Rajeev Thakur, Brian Barrett, P. Balaji, William Gropp, and K. Underwood. Remote Memory Access Programming in MPI-3. ACM Transactions on Parallel Computing (TOPC), Jan. 2015. accepted for publication on Dec. 4th.

[53] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99%. CoRR, abs/1702.04024, 2017. URL http://arxiv.org/abs/

1702.04024.

[54] Ana Klimovic, Yawen Wang, Christos Kozyrakis, Patrick Stuedi, Jonas Pfefferle, and Animesh Trivedi. Understanding ephemeral storage for serverless analytics. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 789–794, Boston, MA, July 2018. USENIX Association. ISBN 978-1-939133- 01-4. URL https://www.usenix.org/conference/atc18/ presentation/klimovic-serverless.

[55] Knative. https://knative.dev/, 2021. Accessed: 2021-11- 21.

[56] Kubernetes. https://kubernetes.io/, 2021. Accessed: 2021-11-29.

[57] AWS S3. https://aws.amazon.com/s3/, 2006. Accessed: 2021-06-07.

[58] Firecracker. https://github.com/firecracker-microvm/ firecracker, 2018. Accessed: 2020-01-20.

[59] Ethan G. Young, Pengfei Zhu, Tyler Caraza-Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. The true cost of containing: A gvisor case study. In 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), Renton, WA, July 2019. USENIX Association. URL https://www.usenix. org/conference/hotcloud19/presentation/young.

[60] LambdaML. https://github.com/DS3Lab/LambdaML/ blob/master/examples/lambda/s3/linear_s3_ga_ma. py, 2020. Accessed: 2020-01-20.

[61] Daniel Barcelona-Pons, Marc Sanchez-Artigas, Gerard Par ´ ´ıs, Pierre Sutra, and Pedro Garc´ıa-Lopez. On the faas track: Build- ´ ing stateful distributed applications with serverless architectures. In Proceedings of the 20th International Middleware Conference, Middleware '19, page 41–54, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450370097. doi: 10.1145/3361525.3361535. URL https://doi.org/10.1145/3361525.3361535.

[62] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, OSDI'18, page 427–444, USA, 2018. USENIX Association. ISBN 9781931971478.

[63] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation, NSDI'19, page 193–206, USA, 2019. USENIX Association. ISBN 9781931971492.

[64] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. Speedo: Fast Dispatch and Orchestration of Serverless Workflows, page 585–599. Association for Computing Machinery,

New York, NY, USA, 2021. ISBN 9781450386388. URL https://doi.org/10.1145/3472883.3486982.

[65] Zhipeng Jia and Emmett Witchel. Nightcore: Efficient and scalable serverless computing forlatency-sensitive, interactive microservices. In Proceedings ofthe 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '21, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3445814.3446701. URL https://doi.org/10.1145/3445814.3446701.

[66] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. SONIC: Application-aware data passing for chained serverless applications. In 2021 USENIX Annual Technical Conference (USENIX ATC 21), pages 285–301. USENIX Association, July 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/ conference/atc21/presentation/mahgoub.

[67] Swaroop Kotni, Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu. Faastlane: Accelerating Function-as-a-Service workflows. In 2021 USENIX Annual Technical Conference (USENIX ATC 21), pages 805–820. USENIX Association, July 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/ conference/atc21/presentation/kotni.

[68] Mike Wawrzoniak, Ingo Muller, Rodrigo Fraga Barcelos ¨ Paulus Bruno, and Gustavo Alonso. Boxer: Data analytics on network-enabled serverless platforms. In 11th Annual Conference on Innovative Data Systems Research (CIDR'21), 2021.

[69] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Submillisecond startup for serverless computing with initializationless booting. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20, page 467–481, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi: 10.1145/3373376.3378512. URL https://doi.org/10.1145/3373376.3378512.

[70] Vojislav Dukic, Rodrigo Bruno, Ankit Singla, and Gustavo Alonso. Photons: Lambdas on a diet. In Proceedings of the 11th ACM Symposium on Cloud Computing, SoCC '20, page 45–59, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381376. doi: 10.1145/3419111.3421297. URL https://doi.org/10.1145/3419111.3421297.

[71] Simon Shillaker and Peter Pietzuch. Faasm: Lightweight isolation for efficient stateful serverless computing. In 2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20), pages 419–433, 2020.

[72] Sol Boucher, Anuj Kalia, David G. Andersen, and Michael Kaminsky. Putting the "micro" back in microservice. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages

645–650, Boston, MA, July 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/ conference/atc18/presentation/boucher.

[73] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. SAND: Towards High-Performance serverless computing. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 923–935, Boston, MA, July 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/ conference/atc18/presentation/akkus.

[74] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. Seuss: Skip redundant paths to make serverless fast. In Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368827. doi: 10.1145/3342195.3392698. URL https://doi.org/10.1145/3342195.3392698.

[75] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid task provisioning with serverless-optimized containers. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 57–70, Boston, MA, July 2018. USENIX Association. ISBN 978-1-931971-44-7. URL https://www.usenix. org/conference/atc18/presentation/oakes.

[76] Yiming Zhang, Jon Crowcroft, Dongsheng Li, Chengfei Zhang, Huiba Li, Yaozheng Wang, Kai Yu, Yongqiang Xiong, and Guihai Chen. Kylinx: A dynamic library operating system for simplified and efficient cloud virtualization. USENIX ATC '18, page 173–185, USA, 2018. USENIX Association. ISBN 9781931971447.

[77] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My vm is lighter (and safer) than your container. In Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17, page 218–233, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132763. URL https://doi.org/10.1145/3132747.3132763.

[78] Ricardo Koller and Dan Williams. Will serverless end the dominance of linux in the cloud? In Proceedings of the 16th Workshop on Hot Topics in Operating Systems, HotOS '17, page 169–173, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350686. doi: 10.1145/3102980.3103008. URL https://doi.org/10.1145/3102980.3103008.

译者跋

原文

Copik 写了一系列关于 serverless 的文章。严肃地说，他这几篇文章（rFaaS、*PraaS*、FMI）核心思想都是差不多，而且没有什么实际用途。但没有实际用途的东西多了去了了，尤其是提出新的 serverless 系统的文章。相比较而言，Copik 的系统还有些新意，并且他在论述自己的优点的时候还比较遵循逻辑。

翻译这篇文章的目的就是为了翻译，手痒了打几个字，如果能方便其他人就再好不过了。

哦还有个原因，我喜欢中文，我喜欢汉字。

（翻译的，不要钱。本文定价0元）