上海交通大学硕士学位论文

无服务器计算范式下高性能DNN训练框架研究

姓　　名：陈星耒

学　　号：122033910115

导　　师：马汝辉教授

院　　系：计算机科学与工程系

学 科/专 业：计算机技术

申 请 学 位：工程硕士

2025 年 2 月 14 日

A Dissertation Submitted to

Shanghai Jiao Tong University for the Degree of Master

RESEARCH ON HIGH-PERFORMANCE DNN TRAINING FRAMEWORKS IN SERVERLESS COMPUTING PARADIGM

Author: Chen Xinglei

Supervisor: Prof. Ma Ruhui

Computer Science and Engineering

Shanghai Jiao Tong University

Shanghai, P.R. China

February 14th, 2025

上海交通大学
学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全知晓本声明的法律后果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

上海交通大学
学位论文使用授权书

本人同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。

本学位论文属于：

□**公开论文**

□**内部论文**，保密□ 1 年/□ 2 年/□ 3 年，过保密期后适用本授权书。

□**秘密论文**，保密____年（不超过 10 年），过保密期后适用本授权书。

□**机密论文**，保密____年（不超过 20 年），过保密期后适用本授权书。

（请在以上方框内选择打“✓”）

学位论文作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

摘 要

无服务器计算范式作为云计算范式下的新形态，以按需计费、弹性伸缩、快速部署等优势成为服务部署的新兴方式。然而，由于无服务器计算范式天然的无状态特性，在迁移分布式深度神经网络训练等有状态应用时，该范式仍面临着一些挑战和局限性。一方面，由于无服务器计算范式的无状态特性，计算函数之间无法进行直接的网络通信。当前的通信通道和通信模式的实现方案带来了较高的通信开销，使得通信阶段成为了整体训练效率的瓶颈。另一方面，尽管无服务器计算范式减轻了开发者对底层资源的部署压力，但是在涉及分布式训练任务时，开发者仍需配置诸多与训练性能相关的系统参数。错误的参数配置会同时导致模型训练性能的衰退和训练开销的上涨。

为了解决上述挑战，本文提出了一个高效的无服务器深度学习训练架构——FasDL，包含以下三部分工作：

1. 本文设计了一种无服务器计算范式下工作节点间的通信模式 K-REDUCE，以减少参数聚合所需的通信时间。K-REDUCE 通信模式能根据模型的训练和通信性能，选择最优数量的工作节点参与聚合，以最小化通信开销。同时，K-REDUCE 通信模式设计了一种混合异步并行（HAP）协议，通过不均等数据划分和异步策略，充分利用非聚合节点在聚合阶段的空闲 CPU 算力进一步加速训练。
2. 本文构建了无服务器计算范式下分布式训练任务性能的轻量级的数学模型，对训练任务的端到端训练时间、训练开销和收敛性能进行建模。通过将训练任务的性能指标与训练负载的配置参数进行关联，本架构可以实现对训练任务性能的良好预测。
3. 本文设计了一种基于剪枝策略的两阶段启发式搜索算法，以实现高效的最优参数搜索。该搜索策略对系统参数的搜索空间进行剪枝，并将搜索参数过程划分了两个独立的阶段，从而在轻微的性能损失下极大提高了参数配置的效率。

为了验证该架构的性能，本文在 AWS Lambda 平台上实现了一个 FasDL 训练框架原型。实验结果表明，在训练性能的预测上，FasDL 的性能建模模块的预测误差控制在 6% 以内。对比 LambaML 的通信模式实现，K-REDUCE 通信模式在训练速度上提升了 16.8%，训练成本降低了 28.3%，同时在大部分情况下取得了更优的收敛性

能。另外，FasDL 训练架构可以兼容不同存储服务的通信通道的实现，相比基于参数服务器的 λDNN，在最优情况下节省了 78.13% 的训练开销。

关键词：无服务器计算，深度学习，通信优化，参数配置

Abstract

The serverless computing paradigm, as a new form under the cloud computing paradigm, has become an emerging approach for service deployment due to its advantages such as pay-as-you-go billing, elastic scaling, and rapid deployment. However, because of the inherent stateless nature of the serverless computing paradigm, it still faces some challenges and limitations when migrating stateful applications like the training of distributed deep neural networks. On the one hand, due to the stateless nature of the serverless computing paradigm, direct network communication between computing functions is impossible. The implementation schemes of current communication channels and communication modes bring about relatively high communication overheads, making the communication stage a bottleneck for the overall training efficiency. On the other hand, although the serverless computing paradigm relieves developers of the pressure of deploying underlying resources, when it comes to distributed training tasks, developers still need to configure many system parameters related to training performance. Incorrect parameter configurations will simultaneously lead to a decline in model training performance and an increase in training costs. To address the above challenges, this paper proposes an efficient serverless deep learning training architecture - FasDL, which includes the following three parts of work:

1. This paper designs a communication mode named K-REDUCE between worker nodes under the serverless computing paradigm, aiming to reduce the communication time required for parameter aggregation. The K-REDUCE communication mode can select the optimal number of worker nodes to participate in the aggregation according to the training and communication performance of the model, so as to minimize the communication overhead. Meanwhile, the K-REDUCE communication mode devises a Hybrid Asynchronous Parallel (HAP) protocol. Through unequal data partitioning and asynchronous strategies, it can further accelerate the training by making full use of the idle CPU computing power of non-aggregation nodes during the aggregation phase.
2. This paper constructs a lightweight mathematical model for the performance of distributed training tasks under the serverless computing paradigm, which models the

end-to-end training time, training cost, and convergence performance of the training tasks. By correlating the performance indicators of the training tasks with the configuration parameters of the training load, this architecture can achieve a good prediction of the performance of the training tasks.

3. This paper designs a two-stage heuristic search algorithm based on the pruning strategy to achieve highly efficient optimal parameter search. This search strategy prunes the search space of system parameters and divides the process of parameter search into two independent stages, thus significantly improving the efficiency of parameter configuration with only a slight performance degradation.

To verify the performance of this architecture, this paper implements a prototype of the FasDL training framework on the AWS Lambda platform. The experimental results show that in terms of training performance prediction, the prediction error of the performance modeling module of FasDL is controlled within 6%. Compared with the communication mode implementation of LambaML, the K-REDUCE communication mode improves the training speed by 16.8%, reduces the training cost by 28.3%, and achieves better convergence performance in most cases. In addition, the FasDL training architecture is compatible with the implementation of communication channels of different storage services. Compared with the λDNN based on the parameter server, it saves 78.13% of the training cost under optimal conditions.

Key words: Serverless Computing, Deep Learning, Communication Optimization, Resource Configuration

目 录

插 图

表 格

算 法

第 1 章 绪论

1.1 研究背景及意义

随着信息技术的飞速发展，云计算已成为支持现代企业运营的核心技术之一。云计算范式通过集中式的数据中心，提供了计算资源的弹性扩展和按需服务，极大地促进了资源的优化利用和成本的降低。随着技术创新和市场需求的不断变化，云计算范式不断演化。最初的云计算范式，基础设施即服务（Infrastructure-as-a-Service, IaaS）[1]，通过提供虚拟化的计算资源，如服务器、存储和网络，使企业能够按需租用硬件资源，减少了物理硬件的投资和维护成本。随后，平台即服务（Platform-as-a-Service, PaaS）[2] 应运而生，它提供了除基础设施之外的运行环境和开发工具，使开发者能够在云端更加方便地开发、运行和管理应用程序。在平台即服务之后，软件即服务（Software-as-a-Service, SaaS）[3] 模型迅速崛起，成为云计算发展的另一重要里程碑。 SaaS 提供了完整的软件解决方案，用户通过互联网访问，无需在本地安装或运行应用程序。这种模式允许用户通过订阅而非购买软件，极大地简化了企业的软件部署和维护工作，同时也使软件更新和升级更加高效。然而，随着企业对于更高效率、更低延迟和更灵活的资源管理的需求日益增长，传统云计算的一些固有局限性逐渐显现。例如，虽然传统模型提供了资源的弹性扩展，但在资源分配和运营成本方面仍存在不足。在这种背景下，无服务器计算范式，或称作函数即服务（Function-as-a-Service, FaaS）作为一种新的计算范式应运而生。

无服务器计算范式[4-6]，亦称作函数即服务（Function-as-a-Service, FaaS），为开发者提供了一个平台，使他们能够将单一功能的代码部署到云平台，并仅在请求时执行。这种模式通过精确的按需启动，显著减少了资源的浪费并提高了应用的响应速度。无服务器计算的核心优势之一在于其高度自动化的资源管理系统。这种系统能够实时监控应用的资源使用情况，并根据需求自动分配或回收资源，从而确保资源的最优利用。例如，当一个函数在执行时，系统会自动为其分配 CPU 和内存资源，一旦执行完成，这些资源会被立即释放，以供其他任务使用。在计费模式上，无服务器计算精确按使用量计费。这意味着用户只需为实际消耗的计算时间和资源支付费用，而不必为持续的资源保留支付不必要的费用。这种计费模式使得成本控制更为精确，尤其适合那些运行间歇性或不连续工作负载的应用。此外，无服务器架构的自动和即时资源扩展功能为应对突发的高并发请求提供了强大的技术支撑。在面

对突然增加的访问量时，无服务器平台可以迅速增加执行实例的数量，以处理这些额外的请求，确保应用的响应速度不受影响。一旦高峰过去，系统会自动减少资源的分配，从而避免无谓的资源浪费。这种能力不仅提升了服务的可靠性和可用性，也使得企业能够更有效地管理运营成本，尤其是在需求波动大的业务场景中。与传统的云服务模型相比，无服务器计算消除了对物理服务器的管理需求，使开发者可以更专注于代码和业务逻辑的优化。此外，无服务器计算支持各种编程语言和框架，提供了与其他云服务和 API 的高度集成，从而为构建复杂、高度可扩展的应用程序提供了强大的灵活性和兼容性。上述无服务器计算范式的特性和优势吸引了许多研究者，关注如何将不同的应用部署在该范式上以提升系统性能，包括视频处理任务[7]、高性能计算任务[8]和深度学习训练任务[9-10]等。

无服务器计算范式中，一个核心的设计原则是函数的无状态性（statelessness）。在这种模型下，函数被设计为独立执行的单元，每次调用都是从零开始，不保留之前的任何执行状态或本地数据。这种设计的主要原因是提高系统的可扩展性和弹性。由于函数不保存状态，它们可以在任何服务器上随时启动和执行，这使得云服务提供商能够在多个服务器和地理位置之间灵活地调度这些函数，以应对不同的负载需求。这种无状态特性也简化了系统的管理和维护，因为每个函数实例都是自包含的，不依赖于特定的物理或虚拟服务器。此外，无状态设计还有助于提高系统的容错性。如果一个函数实例在执行中失败，可以立即在任何其他服务器上重新启动它，而不需要迁移或恢复状态数据。这种快速恢复的能力是无服务器计算在处理高并发和高可用性要求的应用场景中的一大优势。因此，无服务器计算特别适合于无状态的、事件驱动的应用场景，例如自动化任务处理和实时数据分析。在这些场景中，无服务器架构可以实现高效的事件响应，比如自动处理用户上传的数据。企业可以利用无服务器计算来构建自动化的工作流，如图像或视频的即时处理，这些任务通常对启动速度和执行效率有高要求。无服务器平台通过即时分配必要的计算资源，确保了任务的迅速处理，从而优化了应用性能和用户体验。

然而，当涉及到有状态的应用时，如分布式深度神经网络（Deep Neural Network, DNN）训练，无服务器计算面临一些挑战和局限性。由于无服务器平台主要设计用于短期、无状态的计算任务，它通常不保留应用的内部状态，这使得持续的数据处理或长时间运行的任务难以实现。此外，有状态应用如 DNN 训练通常需要持续且复杂的数据交互和状态管。但由于无服务器函数的无状态性，使得这在无服务器环境中难以高效支持。例如，在传统的分布式 DNN 训练中，参数的有效聚合依赖于持续

的网络通信和节点之间的协调，这在无服务器环境中由于缺乏持久连接和状态保持而变得尤为复杂和低效。同时，尽管无服务器计算平台简化了底层硬件的管理，开发者在部署时仍需手动配置诸如计算能力、内存容量以及训练所需的节点数量等资源。这不仅增加了开发的复杂性，也可能导致资源的不充分利用或过度配置，从而影响训练效率和成本效益。因此，虽然无服务器计算为某些应用场景提供了理想的解决方案，但在处理需要复杂状态管理和持续资源利用的任务时，则需要额外的策略和技术解决方案。

本工作旨在探索和开发一种新的分布式 DNN 训练框架，该框架能够在无服务器计算环境下有效支持有状态的应用需求。通过引入高效的函数间通信模式，优化资源配置策略，本研究期望克服现有无服务器计算在支持复杂 DNN 训练任务时面临的技术挑战，从而扩展无服务器计算的应用领域，提升其在实际应用中的性能和可用性。

1.2 国内外研究现状

本节对无服务器计算范式下的分布式 DNN 训练框架相关的国内外研究现状进行介绍。首先，第 1.2.1节对现有的分布式 DNN 训练框架进行介绍。其次，由于分布式训练过程分为训练阶段和通信阶段，因此第 1.2.2分别对当前基于无服务器计算范式的通信优化方案进行介绍。最后，由于无服务器计算范式下的分布式训练任务涉及复杂的参数配置，因此第 1.2.3节对当前无服务器计算范式下的性能预测和参数配置优化相关的工作进行介绍。

1.2.1 分布式深度神经网络训练框架研究

在分布式 DNN 训练中，有三种常见的并行机制以应对训练数据量的增长以及深度学习模型的复杂性，分别是数据并行（Data Parallelism）、模型并行（Model Parallelism）和流水线并行（Pipeline Parallelism）。

参数服务器[11-12] 是数据并行的开创性范例。数据并行在分布式训练环境中，将训练数据划分成多个子集，然后将这些子集分配到不同的计算设备（如 GPU 或 CPU 核心）上进行处理。每个设备使用相同的模型结构和参数初始值，独立地对分配到的数据子集进行前向传播和反向传播计算。 λDNN[13] 延续了基于数据并行的参数服务器实现，并将训练环境迁移至无服务器计算平台上，使用计算函数代替虚拟机成为训练节点。但这种以参数服务器为中心的数据并行方案，其性能在面临大量训练

数据时会受制于参数服务器的实现。为此，后续研究关注与无服务器计算范式下去中心化的数据并行方案。Jiang[14] 提出了一种更高效的分散式规约（scatter-Reduce）通信方法，通过利用无服务器计算函数本身的通信和计算能力实现参数聚合。

模型并行[15-17] 则将一个深度神经网络模型的不同部分（如不同的层或者模块）分配到不同的计算设备上进行计算。其优势在于当模型的规模（如参数数量、层数等）非常大，超出单个计算设备的内存容量时，模型并行可以使训练成为可能。尽管模型并行可以带来加速和内存优化的好处，但随着模型划分的增加和计算设备的增多，设备之间的通信开销会成为一个重要的问题。在模型并行中，不同计算节点上的模型部分在计算过程中需要交换中间结果，这使得在无服务器计算范式中使用模型并行开展训练面临着较高的通信开销。因此，过往研究转而关注于无服务器范式下基于模型并行的推理优化。Duan[18] 等人提出了一种名为 MOPAR 的模型划分框架用于无服务器计算平台上的深度学习推理服务。它基于深度学习推理服务的全局差异和局部相似两种资源使用模式，采用混合方法将深度学习模型垂直划分为多个由相似层组成的切片，对包含资源主导算子的切片进一步划分为多个子切片，以实现并行优化并减少推理延迟。

近来，诸多研究[19-22] 关注在流水线并行的优化上。流水线并行结合了数据并行和模型并行的特点。它将模型划分为多个阶段，每个阶段放置在不同的计算设备上。数据像在流水线上一样依次通过这些阶段，每个阶段对数据进行一部分处理。同时，不同的数据批次可以在不同的阶段并行处理。流水线并行既能够利用多个计算设备的计算资源，又能够在一定程度上减少模型并行中的通信开销。通过让不同的数据批次在不同阶段并行处理，实现了设备的高效利用，加快了训练速度。现有工作开始关注于无服务器计算范式下的流水线并行训练方案。 Funcpipe[23] 提出了收个基于流水线并行的无服务器计算训练框架。该训练框架将训练的通信过程划分为不同子阶段，并设计多个训练节点之间的流水线处理，实现了深度学习模型的快速、低成本训练。 Thorpe[24] 等人提出了一个基于流水线的 GNN 的分布式训练系统。该工作基于配备无服务器计算函数的 CPU 服务器，利用 GNN 的固有特性来分离计算任务，并且将无服务器函数用于轻量级线性代数运算。通过重叠服务器与无服务器函数上的计算任务，该系统有效地隐藏因 Lambda 函数产生的网络延迟。

在无服务器计算范式下，计算函数之间无法直接进行点对点的通信。而模型并行的通信机制相比数据并行更为复杂，且对于计算平台的特性要求更高。因此本文基于数据并行优化无服务器计算范式下的分布式训练框架。

1.2.2 无服务器计算下的通信优化研究

无服务器计算范式的兴起已经引起了深度学习研究的广泛关注[25-27]。由于无服务器计算函数的无状态（stateless）特性，函数间缺乏点对点的直接通信。这导致在无服务器计算平台上开展分布式训练任务时，多个计算函数之间参数聚合的通信开销成为了训练任务的主要瓶颈。为此，诸多研究开始探索如何对函数间的通信机制进行优化。

为了优化分布式 DNN 训练任务的性能，Feng[28] 提出了一个基于数据并行训练大型深度神经网络的无服务器计算架构。该工作通过带有无服务器函数的多层参数服务器（PS）架构减少梯度传输的延迟。但由于函数间缺乏直接通信的通道，该分层式的参数服务器结构给分布式 DNN 训练仍然引入了较大的通信开销。 FaaSFlow[29] 实现了一个自适应存储库 FaaStore，使相同节点上的函数之间能够进行快速的数据传输，从而绕过了外部通道。然后，该实现对底层无服务器计算平台的容器调度提出了额外的限制，因此该面向工作流的优化策略无法有效迁移到分布式训练任务中。 Yu[30] 等人提出了 Pheromone，它配备了一种以数据为中心的函数编排方法，支持函数之间直接且高效的数据交换。该工作将函数间数据交换通道抽象为数据桶，并提供了数据触发原语以同步函数间的数据生成和数据消费，从而实现了函数间的高效交互。 Jiang[14] 等人提出了 LambdaML 框架对无服务器计算范式下的分布式训练流程进行了详细的介绍，包括不同通信模式的对比，如 AllReduce 和 ScatterReduce。同时，该文中对分布式训练过程中工作节点的不同同步策略进行了说明。λDNN[13] 使用独立的参数服务器作为无服务器计算函数之间的通信媒介。该工作指出，无服务器计算范式下的分布式 DNN 训练的性能瓶颈主要来自于参数服务器的资源瓶颈和计算函数较小的本地批量大小。 Funcpipe[23] 则在 LambdaML 中 ScatterReduce 通信模式的基础上，设计流水线并行策略以减少通信阶段的开销。该架构使用模型分区策略来弥补无服务器计算函数的内存和带宽对高性能训练支持上的不足。

此外，一些研究侧重于改进外部存储系统的性能，从而提高无服务器计算函数间的通信效率。 Klimovic[31] 提出了一个弹性的分布式数据存储系统 Pocket。该存储系统支持在 CPU 核心、网络带宽和存储容量等多个维度上动态调整资源，并支持高性能的自动扩展，符合无服务器计算函数通信的特征。 Zhang[32] 等人构建了一个低延迟的多租户云存储 Shredder，能够在存储节点内直接执行小规模的计算。同时，该存储架构为用户提交的 JavaScript 计算函数提供了直接的数据交互能力，使得函数间交互无需通过网络传输数据。主流云服务平台也提供了许多网络存储服务，可以作

为无服务器计算函数间通信的媒介。AWS S3[33] 亚马逊网络服务提供的一种对象存储服务。它有着极高的可扩展性，可以存储海量的数据，并且能够保证数据的可用性、安全性和性能。AWS S3 可以作为 AWS Lambda 函数的事件源，同时，Lambda 函数在执行过程中可以读取和写入 S3 中的数据。 AWS ElasticCache[34] 一种分布式缓存环境，基于云的缓存来提供对数据的快速访问。lastiCache 具有高性能和可扩展性的特点，能够自动扩展和收缩，根据应用程序的负载动态地调整缓存资源，以此确保高性能的缓存服务。AWS ElastiCache 可以作为 Lambda 函数的缓存层，用于提高函数执行的性能。同时，也可以通过事件驱动的方式来更新 ElastiCache 中的缓存。

另一种函数间通信优化的方式是使用 NAT 穿越（NAT-traversal）技术，使得函数之间能够直接实现通信[35]。然而，NAT 穿越通常需要外部服务器，该外部服务器可能会成为通信的性能瓶颈。

本文在 LambdaML 工作的基础上，进一步优化函数间的参数聚合通信模式，并借助现有的高性能存储系统构建高效的函数间通信通道，以降低无服务器计算范式下的通信开销。

1.2.3　无服务器计算范式下的性能预测和参数配置优化研究

由于无服务器计算范式的出现，使得过往诸多应用负载在迁移时的部署方式发生了较大的改变。因此，大量的研究都致力于在这一范式下实现负载性能的可预测性，并尝试优化资源配置。为了对无服务器计算范式下的工作负载的性能进行预测，过往研究聚焦于对无服务器计算平台的特性研究和对具体应用（如分布式 DNN 训练任务）的性能建模两方面。基于对计算平台和工作负载的性能预测，过往工作在不同的云计算范式下提出了对底层资源的配置策略。

针对无服务器计算平台的性能分析，Yu[36] 等人提出了 ServerlessBench，对无服务器计算平台的特性进行基准测试。该工作关注了无服务器计算下的几个特征指标，包括通信性能、冷启动延迟和性能隔离性等，且对主流无服务器计算平台，如 AWS Lambda[37] 和 Open-Whisk[38] 进行了性能评估。

针对分布式训练任务的性能建模，λDNN[13] 提出了一个函数资源配置框架。该框架能够对无服务器计算范式下分布式 DNN 训练工作负载的性能提供可靠的预测。同时，该工作基于对参数服务器网络带宽和函数 CPU 利用率的分析，构建了一个轻量级的分布式 DNN 训练性能模型，可以对 DNN 训练任务进行高效的资源配置，以优化配置函数的开销。 Funcpipe[23] 基于其提出的无服务器流水线训练框架，设计了一种微批次调度策略，能够通过混合整数二次规划实现对无服务器资源的高效配

置和资源约束条件的满足。 Zheng[39] 等人提出了一个高效的资源配置框架 Cynthia，以提供可预测的分布式 DNN 训练性能，并较少训练开销。该工作设计了一个基于工作节点和参数服务器资源消耗的轻量级分析性能模型，以预测 DDNN 训练时间。通过对训练性能的精准预测，Cynthia 能够在保障训练性能的同时最小化训练预算。Shang[40] 等人提出了一个具备异构感知能力的实例配置框架 spotDNN，能为云环境中的分布式 DNN 训练任务提供可预测的性能。该工作构建了异构集群中分布式 DNN 训练的性能分析模型，并通过轻量级的工作负载分析，进一步设计了一种高性价比的实例配置策略。 Carreira[41] 等人提出了一个端到端的无服务器机器学习框架 Cirrus。该框架将无服务器计算接口与无服务器基础设施有效结合，并部署了一个云虚拟机作为参数服务器，有效地减轻了开发者在无服务器计算平台上部署机器学习工作负载的工作量。 Saha[42] 等人提出了一个无服务器计算范式下的函数内存管理系统 EMARS，能够根据应用程序工作负载的使用情况有效地限制无服务器计算函数的内存大小。同时，该工作还提出了一个可预测的内存管理模型，以优化函数内存的分配。

除了分布式训练任务外，大量的工作[43-45] 致力于优化无服务器框架内，分布式推理任务的性能和资源利用效率。此外，一些研究[46-48] 已经开始对通用的无服务器工作流任务的性能表现和运行成本进行建模。

本文针对无服务器计算范式下的分布式 DNN 训练任务，分别对无服务器计算平台和训练负载的性能进行分析和建模，以提供对整体训练性能的预测能力。同时，本文对以 LambdaML 为代表的去中心化的无服务器训练任务的参数配置优化进行研究，以填补过往工作在该领域的空白。

1.3 本文研究思路

基于上文的调研，本文设计并实现了一个高效的无服务器深度学习训练架构 FasDL，旨在提高在无服务器计算平台上训练深度学习模型的性能，并实现资源参数自动配置以减轻开发者的配置负担。

由于无服务器计算的无状态特性导致缺乏点对点通信能力[4]，阻碍了传统通信模式的实现，例如依赖于工作节点之间稳定连接的环形 AllReduce 和去中心化并行[49]。尽管 LambdaML 通过使用外部存储作为通信通道，并采用 AllReduce 和 ScatterReduce 两种通信模式以应对这一挑战。然而，其通信开销最多可以达到计算延迟的 6 倍[23]。为此，本文重新审视了 LambdaML 中的 AllReduce 和 ScatterReduce 通信

模式的设计，比较了随着聚合节点数量从 1 增加到与工作节点数量相同时的通信性能。基于观察结果，本文设计了一种自适应通信模式，称为 K-REDUCE，从所有工作节点中选择最佳 K 个聚合节点进行模型聚合，以确保通信质量。本文观察到在聚合步骤中，非聚合节点的工作节点处于空闲状态。因此，本文设计了一种不均等的数据集划分方案和一种混合异步并行（HAP）协议，以利用非聚合节点的空闲 CPU 算力进一步加速训练。

此外，无服务器计算虽减轻了运维负担，但深度学习开发者仍需配置系统级和应用级参数，以平衡训练延迟、模型准确性和成本[50]。资源参数的错误配置可能导致训练缓慢或成本高昂。为此，本文构建了无服务器计算范式下，端到端的训练性能与系统参数之间的关系，用于预测任意参数配置下的性能。具体的，本文构建了数学模型，以训练时间、收敛效率和训练开销性能指标，描述系统参数配置与整体性能之间的关系。随后，本文设计了一种基于剪枝的启发式搜索算法，以高效解决配置优化问题。

至此，本文设计并实现了一个高效的无服务器深度学习训练架构，解决了高通信开销和资源配置困难两大主要问题。

1.4　文章结构

本文余下内容的组织结构如下：

- 第 2 章对本文的技术背景和研究动机进行说明。首先，该章对无服务器计算范式下的分布式 DNN 训练任务的通用执行流程进行介绍，并对过往工作中存在的不足和挑战进行分析。接下来，该章对高通信开销和参数配置困难两大问题进行详细分析，通过观察实验说明上述问题对分布式训练任务性能的影响，并确定优化方向。
- 第 3 章对 FasDL 训练框架的整体设计进行介绍。首先，该章对 FasDL 训练框架的模块组成进行综述。然后，该章分别详细介绍了 K-REDUCE 通信模式、系统性能建模模块和参数配置模块的设计，以解决背景部分提出的两大主要挑战。
- 第 4 章对 FasDL 训练框架的实现细节进行说明。该章在 AWS 平台上实现了一个系统原型，并对该实现下的负载分析模块以及 K-REDUCE 训练框架的部署细节进行详细介绍。
- 第 5 章对 FasDL 训练框架的性能表现展开实验分析。具体地，该章对系统性能建模模块的预测性能、参数配置模块的搜索精度和搜索效率、K-REDUCE

通信模式的优化性能进行测量。同时，该章对比了不同通信通道实现方案下，FasDL 训练框架的性能表现。最后，该章将 FasDL 训练框架下的分布式训练性能与单机训练性能进行比较。

- 第 6 章对整篇文章进行了总结，并对未来进一步优化的研究方向进行了概述。

第 2 章 技术背景与研究动机

本章基于绪论研究背景，进一步阐述本文的技术背景与研究动机。首先，第2.1节详细介绍无服务器计算范式下分布式 DNN 训练的工作流程。然后，第 2.2节和第 2.3节指出当前应用无服务器计算范式进行分布式 DNN 训练面临的两大主要问题和挑战，分别是高通信开销与参数配置困难。

2.1 无服务器计算范式下的分布式训练工作流

本节对无服务计算范式下分布式训练的通用流程进行介绍。通过分析训练架构的必要能力，并结合无服务器计算范式的特点，进一步挖掘当前分布式 DNN 训练任务在无服务器计算范式下面临的主要挑战。

对于抽象层面的训练流程，训练任务的对象仍然是 DNN 模型和对应的训练数据集。开发者部署多个训练节点开展分布式训练，每个节点上存储了完整的 DNN 训练模型，并分配了部分训练数据集以进行并行训练。在每一个训练轮次后，各个节点获取了该轮次下的本地模型参数，并与其他所有节点进行通信以完成参数聚合，直到模型参数收敛。

在实际执行过程中，无服务器计算范式下的训练节点是 Lambda 函数。其特性对上述通用流程的迁移产生了阻碍。首先，Lambda 函数是无状态的，其无状态特性体现在有限的运行生命周期、无本地持久存储能力和无函数间直接通信能力三个方面。Lambda 函数是基于事件触发运行的计算模型，且主流无服务器平台实现都对函数的最大运行时间进行了限制。因此，为了满足长时间的 DNN 训练的连续性，训练中需要多次触发 Lambda 函数。另外，Lambda 函数被设计为无状态模型，即每次触发不保留上一次的运行状态。因此在多次函数触发之间，需要借助外部持久存储服务记录中间训练状态。同时，为了实现高并发性能，无服务器计算平台不保证两次任务调用的实际运行实体的同一性。因此，训练过程中训练节点之间无法直接构建参数聚合的通信通道，需要借助外部持久运行节点实现参数聚合。

另一方面，尽管无服务器计算平台承担了底层执行硬件和软件执行环境的部署和管理，开发者仍需要在任务部署中指定具体的资源配置。具体地，在分布式 DNN 训练任务中，开发者需要确定与平台相关和任务相关的参数。前者包括训练节点数量和节点性能（如算力、存储等）等，后者包括训练轮次和训练批次大小等。由于

无服务器计算平台对于函数的资源分配有着特定的规则，因此开发者在为指定 DNN 训练任务分配最优资源参数时面临了更大的挑战。

基于上述分析，本文对无服务器计算平台上训练 DNN 模型的典型工作流程及训练架构所需提供的能力进行了总结和抽象，如图 2.1所示。首先，开发者需要将训练模型和训练数据集进行预处理，并存储在 Lambda 函数可访问的外部持久存储节点中。同时，开发者还需准备一个配置文件，以描述必要的系统级和应用级参数。系统级参数包括无服务器函数的数量及其内存配置，两者都影响着函数的执行性能；而应用级参数则与训练过程更相关，如批次大小。接着，开发者将配置文件通过网关（Gateway）提交给无服务器计算平台下的分布式训练架构。无服务器计算平台并行调用多个函数节点开展训练任务。在分布式训练过程中，由于缺乏点对点通信机制，该架构还应提供一个通信模型以支持函数间的参数聚合。

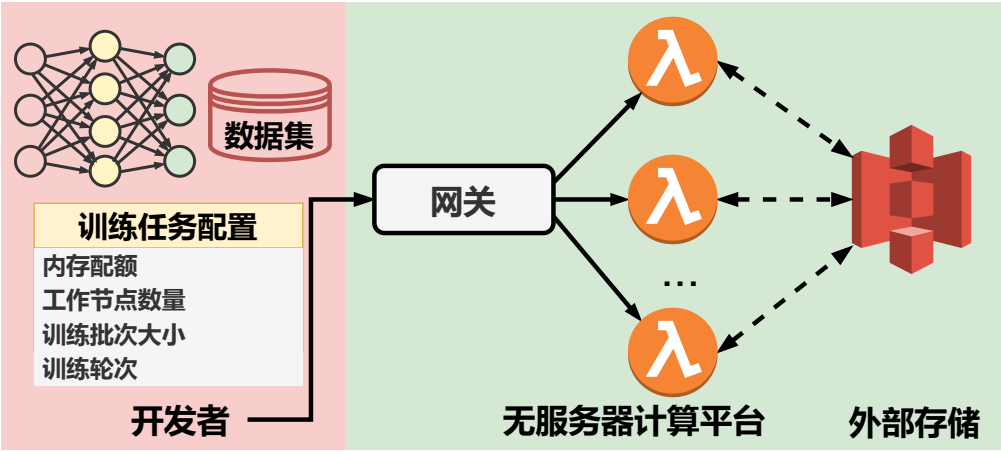


图 2.1　无服务器计算范式下训练DNN模型的典型工作流程及训练架构
Figure 2.1　Typical Serverless-based Distributed Model Training.

在整个流程中，无服务器计算平台下的分布式训练架构面临的核心问题是如何实现一个高效的函数间通信模型，以减少训练中的通信开销。此外，一个高性能的训练架构应提供可预测的模型训练和自动化的参数调优，以提升分布式 DNN 训练性能，并减轻开发者的配置负担。以下针对高通信开销和参数配置困难两大主要挑战进行更具体的分析。

2.2　函数间高通信开销问题

如上文分析，在无服务器架构中，应用被拆分为独立的、事件驱动的函数。这些函数通常在容器或类似的轻量级环境中运行，仅在有需要时被激活。由于每个函

数实例是独立运行，且启动和停止都是动态的，因此在无服务器平台上，函数之间无法直接实现通信。而在分布式训练中，需要多个节点协同工作以共享和更新模型的参数。在这种环境下，缺乏持久的、低延迟的通信通道使得实现有效的参数同步变得复杂。此外，无服务器平台的无状态特性和自动扩展能力虽然带来了管理上的便利，但也增加了维护状态一致性的难度，尤其是在需要频繁交换大量数据的分布式训练场景中。因此，开发者需要采用额外的策略和工具，如引入外部存储解决方案或使用特定的中间件来克服这些通信限制，确保分布式训练的效率和效果。

在无服务器平台上进行分布式训练时，解决函数间通信的问题通常有两种实现策略：基于参数服务器（Parameter-Server Based）的通信通道和基于存储（Storage Based）的通信通道。

基于参数服务器的通信架构是一种常见的分布式机器学习训练方法，特别适用于处理大规模的数据集和模型。参数服务器（Parameter Server, PS）的概念最早由李沐提出[11]。在这种架构中，训练过程被分解成多个工作节点和至少一个参数服务器。工作节点负责处理数据子集，执行本地计算，并计算模型参数的梯度。而参数服务器则负责维护和更新全局模型参数。在每次迭代中，工作节点将计算得到的梯度发送到参数服务器，参数服务器根据这些梯度更新模型参数，并将更新后的参数发送回工作节点。这种方法通过集中式管理模型参数，有效地解决了模型参数在多个节点间同步的问题。然而，这种架构也可能导致通信瓶颈，尤其是在工作节点数量增多时，参数服务器的负载可能会变得非常高，从而影响整体的训练效率。

无服务器计算范式下采用参数服务器作为通信通道开展分布式训练的代表性工作是 λDNN[13]。λDNN 通过启动一个具有固定通信地址的长生命周期服务器作为参数服务器，并启动多个无服务器函数作为工作节点开展训练。无服务器函数能够通过 gRPC 或 HTTP 与参数服务器进行通信，并由参数服务器进行参数的聚合和更新。然而，基于参数服务器实现的通信通道在无服务器计算范式下存在不足。首先，λDNN 指出，随着工作节点的数量增多，参数服务器的网络带宽将成为通信瓶颈。这会导致通信开销的增加，从而影响整体训练性能。其次，这种实现违反了无服务器架构的核心原则——即无需服务器管理和自动扩展。一方面，参数服务器需要开发者部署独立的服务器或使用额外的 PaaS 云服务，这为训练架构的部署增加了额外的配置工作；另一方面，参数服务器缺乏对增加的通信流量的自动扩展支持。且开发者需要根据工作节点的并行数量，调整和配置参数服务器的通信性能。最后，使用参数服务器作为通信通道时，以云虚拟机租用为例，其计费方式是按资源付费。

当不开展训练或训练过程中工作节点进行本地训练时，开发者仍需要为租用的物理资源付费。而无服务器计算范式的核心优势之一是按使用付费（pay-as-you-go）。

以基于存储的通信通道解决函数间通信问题的代表性工作是 LambdaML[14]。LambdaML 架构使用外部持久存储服务，如 AWS S3（Simple Storage Service），作为通信通道，并使用无服务器计算函数，如 AWS Lambda，作为聚合节点进行参数聚合。在该架构下，工作节点除了负责本地训练和参数的传递外，还需负责参数的聚合和更新，并通过外部持久存储服务进行参数的收集和分发。对比基于参数服务器的实现，基于存储的通信通道实现的优势如下。首先，此类存储服务被设计为具有高度可扩展性和高可靠性，且底层通过负载均衡策略确保了高并发场景下的系统性能。其次，此类云存储服务以简易 API 形式提供了快速的读写功能，而无需用户进行额外的部署。最后，尽管不同类型的云存储服务，如对象存储和缓存存储服务，采用了不同的计费方式，但都遵循按使用付费的原则。因此，云存储服务在花销上优于参数服务器。

尽管 LambdaML 提出了一个通用的基于存储服务的通信解决方案，但该训练架构下，通信开销仍然是一个待解决的关键问题。该架构使用工作节点代替参数服务器执行参数聚合的操作，因此，每一轮参数聚合的通信由原本工作节点和参数服务器之间的一次往返，变为工作节点与存储服务之间的两次往返。这将会导致更高的通信开销。FuncPipe[23] 指出，在 LambdaML 架构下开展分布式训练时，其通信时间最高可达到计算时间的6倍。为此，近年来诸多工作聚焦于如何进一步优化通信模式。例如，FuncPipe 提出了一种基于模型并行化的优化方案。该方案通过模型分区和流水线处理来减少通信开销。

本工作洞察到另一个有效的优化方向，即通过调整数据分区和同步策略，重新探索数据并行下通信的并行性，以减少通信开销并进一步加速训练。本工作与 FuncPipe 的优化策略是正交的，为基于存储的通信模式中的高通信开销提供了不同的解决方案。以下将进一步说明本工作的洞察和优化方案的动机。

首先对 LambdaML 提出的基于存储服务的通信通道进行介绍。在基于存储的通信通道的训练架构中，本地训练和参数聚合都由工作节点执行，并采用外部持久存储服务作为通信媒介。各个工作节点在完成了一个轮次的本地训练后，进入到通信阶段借助外部存储服务进行参数聚合，然后再迈入下一个轮次的训练。整个通信过程涉及到无服务器计算函数与外部存储服务直接的两次往返数据传输，如图 2.2所示。在通信过程中，工作节点根据是否参与模型的参数聚合可以划分为两种角色：

聚合节点（aggregator）和非聚合节点（non-aggregator）。根据外部存储服务中存储对象的变化，通信过程可以划分为三个阶段：上传（uploading）、聚合（aggregation）和下载（downloading）。假设有 W 个工作节点，选取其中的 K 个节点作为聚合节点，编号分别为 $1, 2, 3..., K$。在上传阶段，各个工作节点将本地参数以相同的规则分片后上传至外部存储中。分片的数量恰等于聚合节点的数量，同样编号为 $1, 2, 3..., K$。在聚合阶段，各个聚合节点从外部存储中分别拉取对应编号的所有分片到本地，并对对应分片进行参数聚合。然后，各聚合节点再将聚合后的模型参数上传回外部存储服务中。在上述聚合阶段中，非聚合阶段既不执行计算也不参与通信，而是不断轮询以确定聚合是否完成。在下载阶段，各个工作节点从外部存储服务中拉取所有聚合参数分片，并在本地将各参数分片重组为模型参数。特别地，聚合节点无需下载其对应编号的参数分片。由于深度神经网络通常由多层神经元组成，因此为了在上传和下载阶段实现模型参数的划分和重组，需要引入恰当的序列化和反序列化方法，以确保参数聚合的正确性。

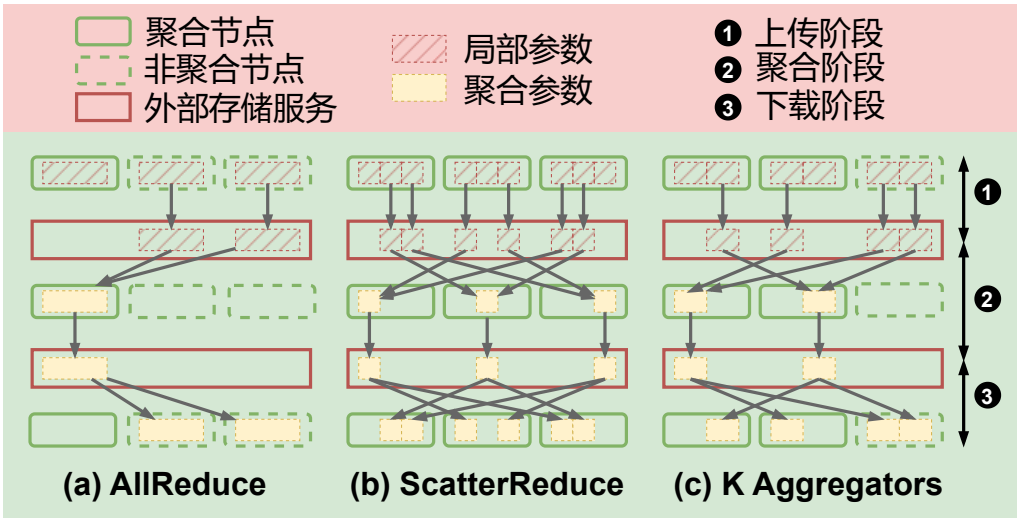


图 2.2 三种基于存储的通道的无服务器通信模式

Figure 2.2 Three Serverless Communication Patterns under Storage-Based Channels

LambdaML 中提出了两种无服务器计算范式下分布式训练的通信模型，分别为 AllReduce 和 ScatterReduce。AllReduce 和 ScatterReduce 是分布式模型训练中常用的方法，图 2.2(a)和图 2.2(b)分别展示了其在无服务器计算范式下的通信流程。 AllReduce 选择 1 个工作节点作为聚合阶段进行参数聚合。其他所有工作节点（即非聚合节点）将其中间参数上传到外部存储，然后聚合节点从外部存储中收集这些参数并进行聚合。最后，聚合节点将合并后的参数存储在外部存储中并分发给其他工作节

点。LambdaML 指出，AllReduce 中选择的一个聚合节点本身的性能会成为参数聚合阶段的瓶颈。为此，其进一步提出了 ScatterReduce 通信模式，增加了聚合节点的数量，从而提高了通信过程的并行度。在 ScatterReduce 中，所有 W 个工作节点都参与聚合阶段，模型参数也被分割成 W 份。在聚合阶段中国，由于所有工作节点并行地从外部存储中下载和上传分片，因此有效地加速了参数聚合的过程，从而解决了单个聚合节点的通信瓶颈问题。

ScatterReduce 的优化思路是符合直觉的，然而本工作在进行深入探究时发现，ScatterReduce 并非在任何场景下都是最优的选择。从本质上来说，AllReduce 和 ScatterReduce 的区别在于聚合节点的数量分别为 1 和 W（即所有工作节点的数量），因此可以看作是聚合过程中并行度的两个极端。然而，过高的并行度在某些场景下会导致通信时间的进一步加长。为了验证上述结论，本工作首先开展了以下观测实验。首先，选择任意数量 K（$1 \leq K \leq W$）的聚合节点对上述两种模式进行泛化。此时对应的通信流程如图 2.2(c)所示。聚合阶段选择了 K 个聚合节点进行参数聚合，并相应地将模型参数分成 K 份。后文默认将聚合节点的数量作为聚合阶段通信并行度的表征，越多的聚合节点数量意味着更高的并行度。其次，为了观察聚合阶段并行度对通信开销的影响，本节在 AWS Lambda 平台上开展了观测实验，选择对象存储服务 AWS S3 作为通信通道。为了研究对不同复杂性模型的影响，本节选择了 ResNet50[51] 和 SqueezeNet[52] 作为训练模型。在一次完整的训练轮次中，观测实验以步长 1 从 1 到 W 设置不同数量的聚合节点进行参数聚合，并统计了训练和通信时间的变化情况，结果如图 2.3和图 2.4所示。这里选择的工作节点数量 W 为 10 个，因此，当 K 等于 1 和 10 时，分别代表了 AllReduce 和 ScatterReduce 的性能。

对于 ResNet50 模型，从图 2.3中可以看出，ScatterReduce 相比 AllReduce 显著降低了通信阶段的时间，与上文分析一致。然而，ScatterReduce 并非最优情况。随着聚合节点数量 K 的增加，通信时间开销的变化趋势呈现出一个“U”字形。初始时，通信开销随着并行度的增加快速下降，并在选择 7 个聚合节点时达到最低。随着并行度的继续增加，通信开销反而出现了少量的回升。该实验表面，一味提高聚合阶段的并行度并不总能带来通信开销上的优化。而对于 SqueezeNet 模型，如图 2.4所示，其呈现出与前文分析相反的结果。通信开销随着并行度的提升持续上升。此时，AllReduce 成为了最优情况。

上述观测实验中，AllReduce 和 ScatterReduce 在 ResNet50 模型和 SqueezeNet 模型上分别表现出了最坏的通信性能，且在最坏情况下，一个训练轮次中的通信训

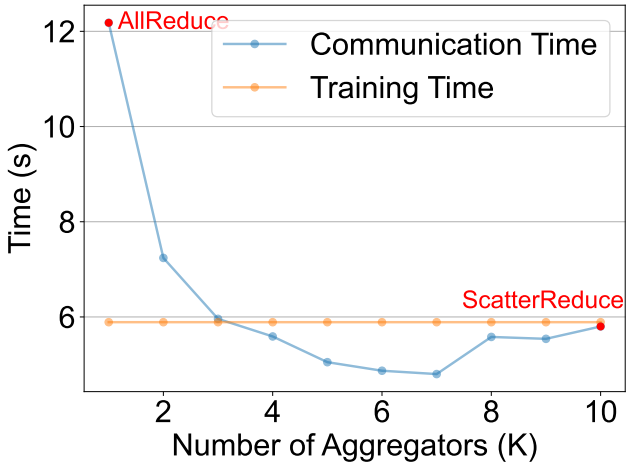


图 2.3 ResNet50 模型训练和通信时间随聚合节点数量变化的趋势

Figure 2.3 The trend of training and communication time of the ResNet50 model with varying numbers of aggregators

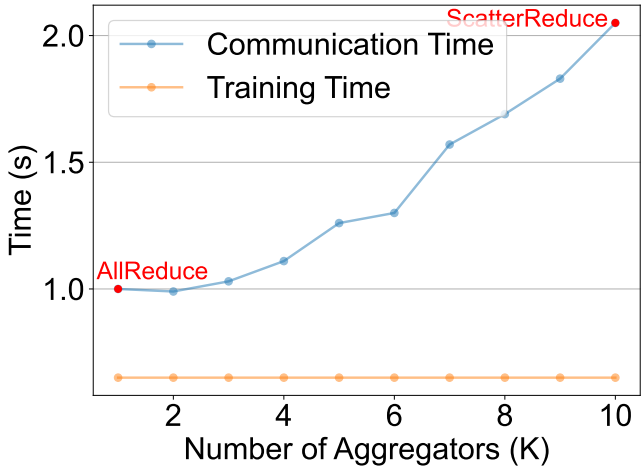


图 2.4 SqueezeNet 模型训练和通信时间随聚合节点数量变化的趋势

Figure 2.4 The trend of training and communication time of the SqueezeNet model with varying numbers of aggregators

练比高达 2 至 3 倍，这显著地影响了分布式训练任务端到端的训练性能。因此，AllReduce 和 ScatterReduce 两种通信模式并不能很好地解决无服务器计算范式下分布式 DNN 训练高通信开销的问题。此时，需要一个能够自适应地根据训练模型特点最优化通信开销的通信模式和训练框架，以减少训练中的通信训练开销比。

2.3　系统参数配置困难问题

无服务器计算范式虽然减少了开发者的运维开销，但深度学习从业者仍然需要配置一些系统级别和应用级别的参数，以最优化训练性能和训练成本。首先，开发者需要向无服务器计算平台申请运行函数作为工作节点，此时，需要配置函数的硬件资源，包括 CPU 核心数量[53-54]、内存配额[55]和网络带宽[56]等。这些参数从不同的方面对训练的性能产生了影响。CPU 核心数量代表了函数的计算能力，越多的算力将提升训练阶段的计算速度。内存配额则需要满足模型的最低运行要求。在深度神经网络训练的过程中，内存需要完整存储深度模型以及前向和后向传播中产生的中间结果。而函数的网络带宽则影响了参数聚合节点的通信延迟。在现有的主流商业无服务器计算平台中，上述函数资源的配置是高度耦合的。以AWS Lambda 平台为例，开发者只能配置 Lambda 函数的内存配额，其配置区间为 1MB 至 10240MB。而 CPU 算力和网络带宽则基于用户定义的内存配额自动进行线性分配。这种高耦合的资源配置方式，为深度学习开发者最优化训练性能带来了更大的挑战。其次，除了函数本身的性能外，开发者还需要指定工作节点的数量。数据并行的策略本身是通过高并行的工作节点以加速训练过程，然而在无服务器计算范式的特性，使得选择最优的工作节点数量变成了一个更为复杂的问题。因为除了考虑训练阶段的影响，工作节点的数量也会影响到通信阶段参数聚合的效果。因此，开发者需要选择合适的工作节点数量，在训练开销和通信开销之间取得最优平衡以最小化端到端的训练任务开销。最后，开发者还需要配置一些应用级的超参数，如训练中的批次大小（Batch Size）等。在不同的系统级参数配置下，应用级参数的选择也会对训练的性能产生不同的影响。在训练阶段，批次大小会直接影响到训练的速度以及收敛性能。

除了训练性能外，训练成本也是开发者需要考虑的重要因素。无服务器计算平台采用了按使用付费的计费模式。这里的使用指的是对 Lambda 函数的使用，包括了函数的资源配置（如内存配额）、函数的数量以及函数的运行时间。由此可见，训练任务的总体开销也与上述参数关联。如何在训练性能和训练开销之间取得最优平衡成为了开发者进行参数配置时的挑战。

为了验证上述猜想，本节在 AWS Lambda 上开展观测实验。该实验使用三组随机选取的参数配置在 ResNet50 模型上训练了相同数量的训练样本，并统计了整个训练过程的端到端训练时间以及训练成本。实验中配置了以下三个参数，包括 Lambda 函数的内存、数量以及批次大小。通信阶段分别采用了 AllReduce 和 ScatterReduce

两种通信模式。

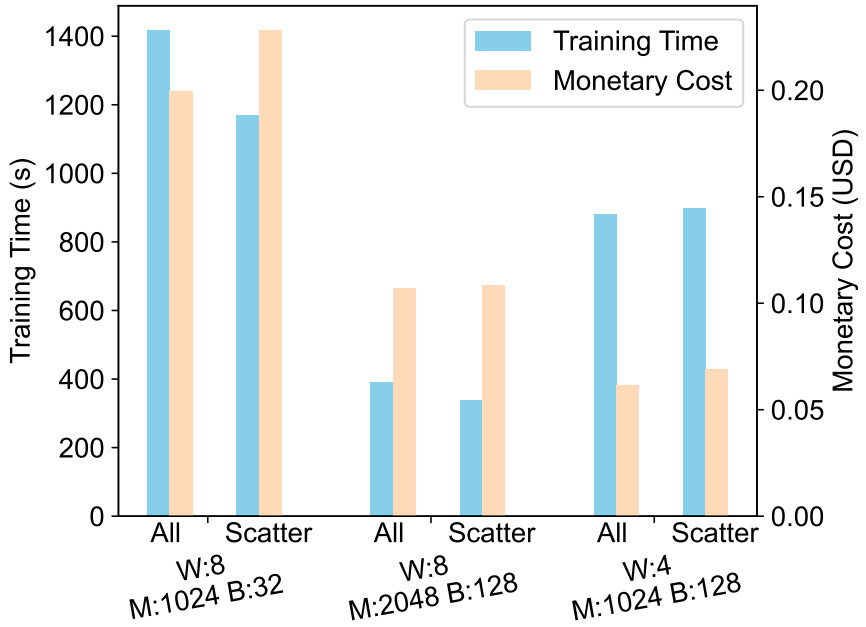


图 2.5 在无服务器计算平台中，不同参数配置（工作节点数 W、内存配额 M 和批次大小 B）下的训练时间和训练成本比较

Figure 2.5 Comparison on Training Time and Monetary Cost in Serverless Environment with Different Configurations (Number of Workers *W*, Memory Quota *M* and Batch size *B*)

如图 2.5所示，使用 8 个工作节点、1024MB 内存和 32 的批次大小进行训练所需的时间和成本是使用 8 个工作节点、2048MB 内存和 128 的批次大小的两倍以上。而后者与使用 4 个工作节点、1024MB 内存和 128 的批次大小的配置在时间和成本上的表现正好相反，即训练时间翻倍但训练成本减少至原来的一半。另外，从第一组配置下的 ScatterReduce 和 AllReduce 对比中能看出，不同的通信模式在相同的参数配置下可能表现出训练时间和训练成本上不同的优势。而从第二组配置和第三组配置下的 ScatterReduce 和 AllReduce 对比中能看出，不同参数配置下两种通信模式也可能在两个指标上均优于另一者。这意味着训练任务的性能与参数配置高度相关，且会受到通信模式的影响。因此，要在特定优化目标下设置最优的参数配置和通信模式变成了一个高度耦合的优化问题。开发者在系统级和应用级参数上的错误配置可能导致不可预见且过高的训练时间和训练成本。这严重阻碍了开发者在无服务器计算平台上部署分布式 DNN 训练任务。

2.4　本章小结

本章节对技术背景与研究动机进行了详细的介绍。首先，第 2.1节对无服务器计算范式下分布式训练的通用流程进行了说明。其中，支持多个工作节点执行参数聚合的函数间通信模型和训练任务的参数配置这两个环节是阻碍训练性能提升的主要挑战。为此，第 2.2节和第 2.3节分别围绕高通信开销和参数配置困难两大挑战，对过往工作的解决方案进行了观察实验，并分析了其不足之处，为下一章节中本文的系统架构设计提供理论支撑。

第 3 章　无服务器计算范式下分布式训练架构设计

基于上一章节中对无服务器计算范式下分布式训练面临的两大关键挑战的分析，本章提出了一个高性能的训练架构 FasDL。该架构设计了一种高效的函数间通信模型 K-REDUCE，有效降低了分布式训练中参数聚合引入的通信开销。同时，该架构对分布式训练的性能进行建模，并基于此提出了一个两阶段启发式参数搜索算法，实现了高效的系统参数配置。

3.1　系统架构及工作流程概述

为了应对无服务器计算范式下分布式 DNN 训练任务面临的高通信开销和参数配置困难问题，本工作提出了一个无服务器计算范式下的高性能深度学习训练架构 FasDL。图 2.4展示 FasDL 的系统架构组成以及训练工作流程。该系统架构由四个主要模块组成，分别是性能建模（Modeling）模块、负载分析（Profiler）模块、参数配置优化（Optimizer）模块和 K-REDUCE 通信模式。

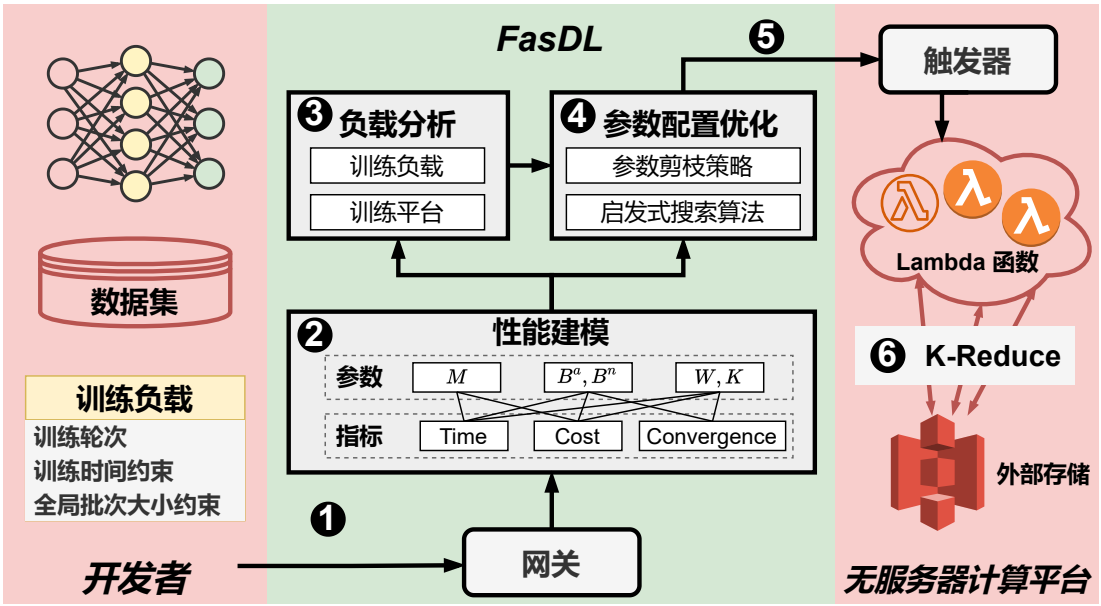


图 3.1　FasDL 系统架构及工作流程

Figure 3.1　System Architecture and Workflow of FasDL

性能建模模块旨在对模型训练任务的性能进行数学化建模，以提供可预测的模型训练。对深度学习开发者而言，其关心的训练指标包括训练任务的端到端时间、

训练的成本以及训练的收敛效果。而上述整体性能指标与训练的参数配置紧密相关，如函数的内存配额、作为工作节点的函数的数量、训练的批次大小以及聚合阶段的聚合节点的数量等。负载分析模块旨在对训练模型和无服务器计算平台的特征进行预分析，作为建模结果的支撑。借助负载分析模块，该架构能够轻易地扩展到不同的训练负载和无服务器计算平台上，为系统提供了良好的可扩展性和自适应能力。参数配置优化模块旨在自动化搜索最优参数配置。首先，该模块基于建模和分析结果，构建特定的优化目标。然后，该模块设计一个基于剪枝策略的两阶段启发式搜索算法，在提高参数配置的效率的同时确保参数配置的结果的有效性。 K-REDUCE 通信模式是本训练架构的核心。该通信模式通过选择最优的聚合节点数量、设计高效的节点同步方案和调整训练数据划分方式，达到了优化分布式训练中参数聚合的通信开销的目的。

FasDL 训练架构的整体工作流程如图 2.4所示。首先，开发者需要通过网关提交其待训练的工作负载，包括训练模型（DNN Model）、训练数据集（Dataset）、训练的总轮次（Epochs）以及相关约束条件。然后，性能建模模块对上述工作负载进行性能建模，并将建模结果传递至负载分析模块和参数配置优化模块。负载分析模块将在工作负载层面和无服务器计算平台层面分别进行性能分析，拟合出对应的性能系数，并将结果传递至参数配置优化模块，作为其开展参数配置搜索的支撑。然后，参数配置优化模块对参数搜索空间进行剪枝以缩小参数搜索范围。进一步地，参数配置优化模块基于两阶段的启发式搜索算法求取当前优化问题的最优参数配置，并将结果传递至无服务器计算平台开展训练任务。分布式训练任务的入口是一个触发节点函数 *Tirgger*，其负责根据参数配置触发训练任务。具体地，该触发器会将开发者上传的数据集按照工作节点的角色进行不均等划分，然后触发对应数量的 Lambda 函数作为工作节点。接下来，各工作节点将借助指定的外部存储服务开展分布式训练。在参数聚合过程中，各工作节点依据 K-REDUCE 通信模式执行参数聚合。为了进一步加速训练，FasDL 架构采用了一种称为混合异步并行协议（Hybrid Asynchronous Parallel, HAP）的同步策略。当训练达到指定轮次后自动结束，并将训练结果上传至指定的外部存储服务中。

3.2　函数间通信模式和同步策略设计

第 2章中分析了现有的两种通信模式 AllReduce 和 ScatterReduce 无法有效应用于不同特点的深度训练模型，进而会导致过高的通信开销。为此，本文设计了一个

称为 K-REDUCE 的通信模式，以提升基于无服务器计算范式的分布式模型训练的性能。其工作流程可以划分为三个阶段。首先，工作节点按照其在聚合阶段的责任划分为两种角色，分别称为聚合节点（aggregator）和非聚合节点（non-aggregator）。第 2.2节的观测实验指出，训练中的通信开销与聚合节点数量的选择有关。因此，K-REDUCE 通过对训练任务的建模自适应地选择最优的聚合节点数量以最小化通信开销。其次，与传统在工作节点间均匀划分训练数据不同的是，K-REDUCE 采用了一种不均等的数据划分策略。，据工作节点的角色不同为其指派不同的任务以及相应数量的训练数据。最后，在分布式训练过程中，本文设计了一种称为混合异步并行协议（Hybrid Asynchronous Parallel, HAP）的同步策略。该策略充分利用了通信过程中非聚合节点的空闲算力资源，以进一步加速整个端到端的训练速度。

3.2.1 最小化通信开销的聚合节点选择

如第 2.2节观测实验结果（图 2.4和图 2.3）所示，不同的训练模型在 AllReduce 和 ScatterReduce 两种通信模式下展现出了截然相反的通信表现。因此，最小化指定训练模型参数聚合的通信开销需要相应选择不同数量的聚合节点。该最优点的选取和训练模型的特性有关。通常来说，增加聚合节点的数量可以提高参数聚合阶段的并行度，从而减少通信开销。然而，在上传阶段和下载阶段中，各个工作节点需要把本地训练后得到的模型参数按照聚合节点数量划分为对应数量的参数分片，然后以外部存储为媒介传送给对应的聚合节点。因此，随着聚合节点的数量的增加，工作节点与外部存储之间的通信次数也随之增加，并且单个参数分片的大小随之减少。

为了验证模型分片大小对参数聚合中传输效率的影响，本节测试了 AWS Lambda 和 AWS S3 之间上下行的吞吐量（throughput）性能，结果如图 3.2和图 3.3所示。具体地，实验在 AWS Lambda 函数中调用 AWS S3 的上传对象（`put_object`）和下载对象（`get_object`）接口向 S3 中上传和下载不同大小的分片，并对接口的吞吐量进行统计。另外，由于 Lambda 函数的网络带宽和用户分配的内存配额直接相关，因此实验在不同内存配额的 Lambda 函数上开展了上述实验。从图中可以看出，随着传输分片的大小增大，上行和下行的吞吐量也随着增大。对特定内存配额的函数，当分片大小超过某个临界值时，其传输吞吐量趋于一个稳定值，即当前内存配额下函数对应的网络带宽。总体而言，内存配额越大，函数的网络带宽能力越强，这与 AWS Lambda 对函数资源的分配规则一致。

上述实验表明，当传输的参数分片过小时，Lambda 函数的网络带宽并没有被充分利用，进而将导致传输速率的下降和通信开销的上涨。特别是对于 SqueezeNet 这

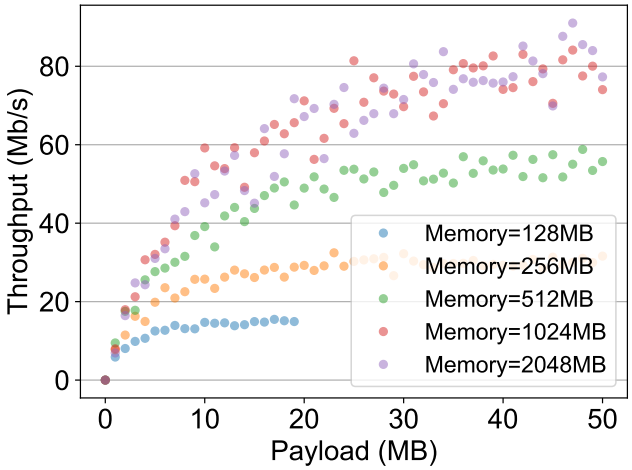


图 3.2　Lambda函数与S3存储服务之间的上行吞吐量
Figure 3.2　Upload Throughput between Lambda Functions and S3

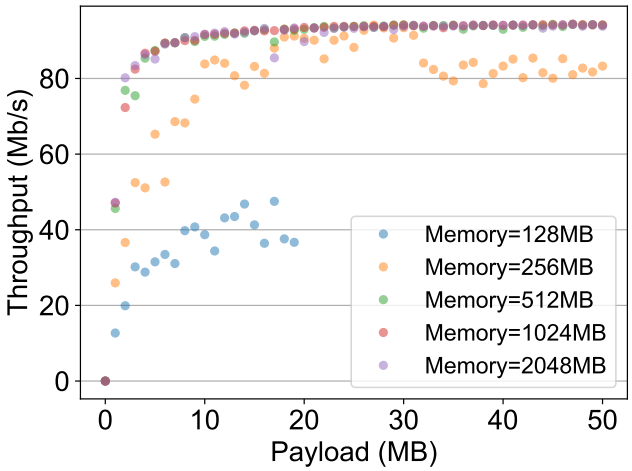


图 3.3　Lambda函数与S3存储服务之间的下行吞吐量
Figure 3.3　Download Throughput between Lambda Functions and S3

样的简单模型，其参数大小只有 4.71 MB。此时，过度的分片导致的传输速率的损失将远大于提高聚合阶段并行度带来的增益。因此，从图 2.4中可以看出，此时最优的通信模式是仅采用一个或两个工作节点作为聚合节点。此后增加新的聚合节点，通信过程的开销都不断的增大。而对于 ResNet50 这样更为复杂的模型，适当提高聚合阶段的并行度将带来显著的收益。但其收益逐渐减小并在某个聚合节点数量的位置达到临界。继续增加聚合节点的数量将使通信开销进一步增大。如图 2.3所示，最优的通信开销出现在聚合节点数量为 7 时。图中“U”字形的通信开销变化曲线是上述

两个影响因素相互作用的结果。

综上，最小化通信开销需要在聚合阶段的并行度和参数分片对传输速率影响之间取得最优平衡，即根据训练模型的特征自适应地选择最优数量的聚合节点参与聚合阶段的通信。为此，第 3.3节基于对训练模型的性能分析，构建了通信开销与聚合节点数量之间的关联。同时，第 3.4节设计了一种高效的启发式算法搜索最优的聚合节点数量。

3.2.2　基于混合异步并行协议的同步策略

如上文所述，K-REDUCE 训练框架中并非所有工作节点都参与到参数聚合的过程中。因此，在参数聚合阶段，工作节点根据其任务的差异划分为两种角色：聚合节点（aggregator）和非聚合节点（non-aggregator）。整个训练过程可以分为本地训练和参数聚合的通信两个部分，且两部分交替进行。其中，通信阶段可以进一步划分为三个阶段，分别是上传、聚合和下载。上传阶段中，各工作节点向外部存储服务上传本地训练后得到的局部模型参数。下载阶段中，各工作节点从外部存储服务中下载聚合后的全局参数模型。因此，所有工作节点都参与了上述两个阶段。而在聚合阶段，仅由选定的聚合节点执行聚合操作。具体地，各聚合节点将其负责聚合的参数分片从外部存储中拉取到本地执行聚合操作，再将聚合后的参数分片回传到外部存储服务中。

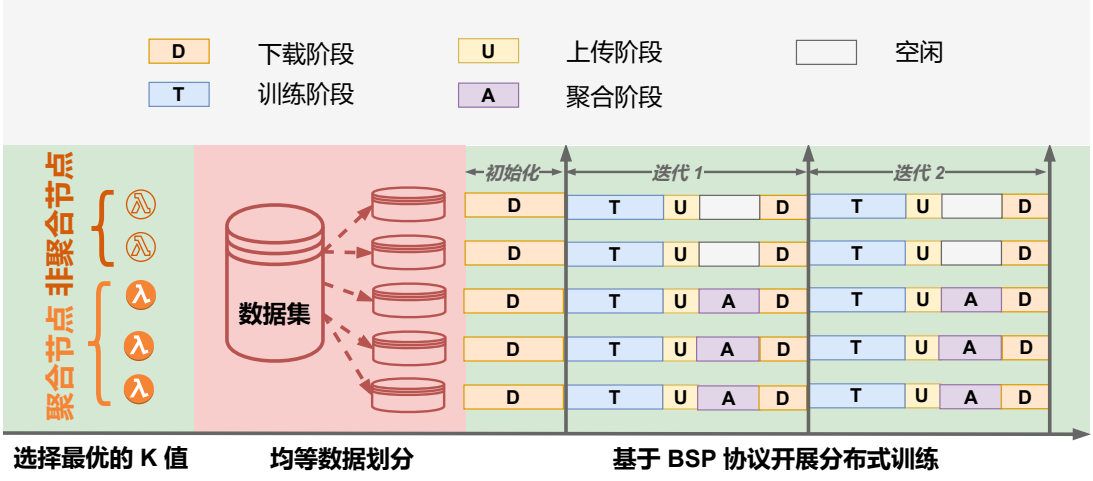


图 3.4　采用数据集均等划分和批同步并行协议的训练工作流程
Figure 3.4　Workflow of Training with Equal-Dataset-Partitioning and BSP Protocol

上述训练过程与 ScatterReduce 和 AllReduce 一样遵循严格的批同步并行协议（Bulk Synchronous Parallel, BSP）[57-58]，以保证多个工作节点之间模型参数的一致

性。BSP 由 Leslie Valiant 在 1990 年提出，其核心思想是将计算过程分为多个阶段，每个阶段包括计算、通信和同步。BSP 的基本单位是超级步骤（super step）。在每个超级步骤中，所有处理器执行本地计算，然后进行通信（发送和接收消息），最后进行全局同步。每个超级步骤的执行是独立的，所有处理器在这一阶段结束时必须达到同步点。这意味着在超级步骤结束时，所有处理器必须等待其他处理器完成其计算和通信。这种同步确保了数据的一致性，使得后续的超级步骤可以安全地使用更新后的数据。

在无服务器计算范式下，采用 BSP 的训练流程如图 3.4所示。超级步骤在该语义下代表了一个完整的迭代（iteration）。在每个迭代的最后，所有节点都会拉取到最新的模型参数，再进入到下一个迭代的训练中。然而，从图中可以看到，每一个迭代中的非聚合节点在聚合阶段都处于空闲状态。此时非聚合节点既不进行训练计算，也不参与任何网络通信，而是不断轮询直到其他聚合节点完成参数聚合。

BSP 是一种同步模型，提供了一种简单且直观的编程模型。其后，许多研究开始关注于异步模型在分布式训练中的应用。Qirong Ho 在其工作中提出了一种名为过时同步并行（Stale Synchronous Parallel, SSP）[59]的并行计算模型。SSP 引入了过时参数（stale parameter）的概念，即在 SSP 模型中，处理器可以使用过时的模型参数进行计算。这意味着每个处理器在执行计算时可能使用的是来自上一个超级步骤的参数，而不是最新的参数。Ho 在基于参数服务器的分布式训练中应用了 SSP 协议，允许工作节点在不完全同步的情况下进行训练计算，从而减少等待时间和通信开销。在其工作中，其提出了一个过时度（staleness）的概念，定义为工作节点获取的模型参数所在的迭代与当前迭代之间的绝对差值，以表征其模型参数的过时程度。开发者将定义一个全局的最大过时度。在该过时度范围内，工作节点可以拉取最新的模型参数，并无需等待其他工作节点而继续开展训练。当且仅当最快的工作节点与最慢的工作节点之间迭代的差值大于该全局最大过时度时，前者需要等待后者。作为一种异步模型，SSP 可以显著减少由于等待其他工作节点而导致的空闲时间，提高整体的计算效率，特别是在处理大规模数据集时。但是，使用过时参数可能会导致模型收敛速度降低，甚至在某些情况下影响最终的模型性能。Ho 在其论文中指出，通过设置恰当的过时度，可以有效控制异步对模型收敛的影响。通常来说，越小的过时度对收敛性能的影响越小。BSP 等同步模型可以看作是过时度为 0 的异步模型，即对应了异步模型在收敛性能上的上界。通过定义恰当的全局过时度，可以在效率和一致性之间取得最优平衡，从而从整体上提升模型训练的效率和性能。

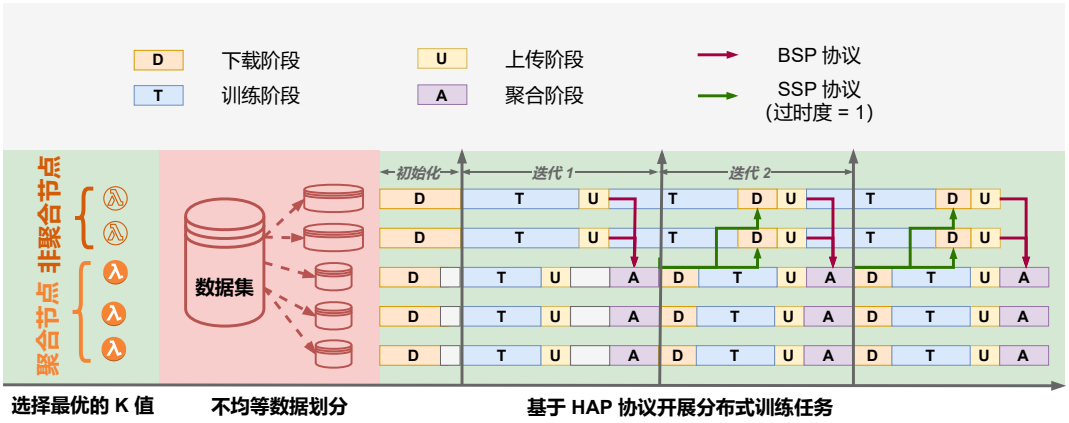


图 3.5　无服务器计算范式下分布式训练K-REDUCE通信模式的工作流程
Figure 3.5　Workflow of K-REDUCE for Serverless-based Distributed Training

基于此，本文提出了一种混合异步并行（Hybrid Asynchronous Parallel, HAP）协议，如图 3.5所示。在 BSP 协议中，非聚合节点在聚合阶段需要轮询等待聚合节点完成本迭代的参数聚合，这使得该时间内非聚合节点的空闲算力资源没有得到充分的利用。因此，HAP 协议允许非聚合节点在上传本迭代的参数分片后，拉取上一个迭代的过时参数，并直接进入到下一个迭代的训练中。具体地，第一个迭代中，非聚合节点仅执行上传阶段并直接进行下一迭代的训练计算。此后每一个迭代中，非聚合节点都会拉取上一个迭代聚合后的模型参数。而对于聚合节点而言，其在第一个迭代中会存在一段空闲等待的状态，以实现与非聚合节点间的同步，其后每个迭代都不再会出现忙等的空闲时间。

在 HAP 协议中，聚合节点之间仍然遵循 BSP 协议同步地开展训练，而非聚合节点则遵循 SSP 协议并保持过时度（staleness）为 1。图 3.5中分别指示了非聚合节点的上传阶段和下载阶段的行为。具体地，非聚合节点异步地拉取上一个迭代中聚合节点更新的全局参数并上传本迭代的本地参数，使聚合节点能够同步地执行参数聚合。尽管异步模型会一定程度上影响模型的收敛性能，但 HAP 协议通过严格约束非聚合节点的最大过时度为 1，并限制聚合节点严格执行同步模型，在保证了整体训练的收敛性能的同时提升了整体训练任务的性能。第 5章对 HAP 协议在系统收敛效率和整体性能上的表现进行了实验验证。

通过指定不同角色的工作节点采取不同的同步策略，HAP 协议充分利用了 BSP 协议中非聚合节点在聚合阶段的空闲时间，以加速模型训练。因此，非聚合节点在同一个迭代中执行训练计算的时间要多于聚合节点。相应地，本文在训练数据划分阶段采用了一种不均等的数据划分方式。非聚合节点分配到了更多的训练数据以使

其更聚焦于模型训练任务。而对于聚合节点则相应减少其训练数据，以使其能够承担通信过程中的参数聚合任务。在初始化节点，聚合节点由于需要拉取的训练数据较少，因此会经历短暂的空闲状态。

如图 3.5所示，聚合节点之间的同步由 BSP 协议保证，非聚合节点之间的同步由 SSP 协议保证，而不同角色的工作节点之间的同步则需要由恰当的训练数据划分保证。对于非聚合节点，其在每个迭代需要执行训练、上传和下载三个阶段。而对于聚合节点，其在每个迭代需要执行训练、上传、聚合和下载四个阶段。因此，为了确保两者之间在迭代（iteration）间的同步以避免引入额外的空闲等待，需要满足每个迭代中非聚合节点与聚合节点的训练时间的差值恰等于聚合节点执行聚合操作的时间。为此，本文为聚合节点和非聚合节点指定了不同的训练批次大小。非聚合节点的批次大小大于聚合节点的批次大小。同时，为了确保两者之间在轮次（epoch）间的同步，需要在训练数据划分阶段依据两者的批次大小等比例地划分训练数据集。第 3.3节中的性能建模模块对训练阶段和聚合阶段的时间开销进行了建模，同时，第 3.4节的参数配置模块将上述分析作为目标优化问题的一个约束条件，以确保最终获取的参数配置能够提供不同角色的工作节点之间在迭代间的同步。在训练开始时，触发节点 *Tirgger* 将依据参数配置结果中两种角色的批次大小和数量，等比例地执行不均等的训练数据划分，以确保其在轮次间的同步。第 4.2节对训练数据划分的实现进行了详细的说明。

3.3　训练性能建模模块

本节对无服务器计算范式下的分布式训练任务的性能进行了建模，并根据 K-REDUCE 通信模式的特点进行了额外的设计和补充。由于 AllReduce 和 ScatterReduce 通信模式可以看作 K-REDUCE 的一种特例，因此该建模结果也适用于上述两种已有的通信模式。

本节的建模考虑了训练性能的三个方面：端到端的训练时间、训练开销计费和收敛效率。端到端的训练时间是分布式训练中最重要的一个性能指标，尤其是当训练数据集的规模非常庞大时。计费是开发者需要向无服务器计算平台支付的资源使用计费，具体包括对无服务器计算函数的运行计费以及对外部存储服务的使用计费。收敛效率表征了训练模型在测试训练集上的收敛情况，是表征训练效果的重要指标。

由于现有的主流商用无服务器计算平台对资源的配置方式都采用了以内存为中

心的耦合分配，即无服务器计算函数的 CPU 算力和网络带宽与内存配额线性绑定。因此，本文依据该资源分配方式展开讨论。对于开发者而言，部署一个分布式训练任务需要指定的参数包括内存配额 M、总工作节点数 W 和批次大小 B。由于 K-REDUCE 通信模式将工作节点进行了分类，因此又额外引入了聚合节点的数量 K ($1 \leq K \leq W$)。同时，由于聚合节点和非聚合节点遵循不同的同步策略，因此需要为两者配置不同的批次大小。后文分别表示为 B^a（batch size of aggregator）和 B^n（batch size of non-aggregator）。

除了上述与训练性能直接相关的配置参数外，建模中还涉及了一些与任务负载和平台性能相关的参数。为了便于理解，表 3.1对后续建模涉及的关键符号进行了总结。

表 3.1　K-REDUCE 性能建模的关键符号
Table 3.1　Key Notation in the Modeling of K-REDUCE

符号	定义
S_m	模型参数大小（Size of the total model parameters）
S_s	参数分片大小（Size of the model parameter shard in transmission）
S_d	训练集大小（Size of the training dataset）
D	训练样本数（Number of training data samples）
I	一个轮次中的迭代次数（Number of training iterations per epoch）
E	训练轮次数（Number of training epochs）
W	工作节点数（Number of provisioned serverless functions as workers）
M	内存配额（Memory allocation of serverless functions）
K	聚合节点数（Number of aggregators）
B^a	聚合节点的批次大小（Batch size of aggregators）
B^n	非聚合节点的批次大小（Batch size of non-aggregators）
B^g	全局批次大小（Global batch size）
tp_{up}	函数与外部存储之间的上行吞吐量（Upload throughput of serverless functions）
tp_{down}	函数与外部存储之间的下行吞吐量（Download throughput of serverless functions）

3.3.1　端到端训练时间建模

如图 3.5所示，端到端训练过程可以拆解为三个部分，分别是初始化阶段、通信阶段和训练阶段。其中，初始化在每次触发函数时执行一次，而通信和训练过程则在后续过程中不断交替。开发者在训练前指定了训练的总轮次（epoch），每一个轮次对训练数据集进行一次完整的训练。其中，每个轮次又分为多轮迭代（iteration），每一次迭代中，各工作节点先训练一个批次的数据，再进入通信阶段执行参数聚合。下文对各个阶段的时间开销进行分析和建模。

3.3.1.1　通信时间建模

如图 3.5所示，通信阶段包括三个阶段，分别是上传、聚合和下载。每个工作节点将模型参数平均分割成 K 个参数分片，并在上传阶段中将它们上传到对应的外部存储服务中。这里 K 为聚合节点的数量。每个参数分片按序与聚合节点对应，由其负责该分片的聚合任务。因此，每个参数分片的大小 S_s 可以表示为式 3-1。

$$S_s = \frac{S_m}{K} \tag{3-1}$$

其中，S_m 为模型参数的大小。

由此，可以计算出上传阶段的用时如式 3-2所示。

$$T_{up} = \frac{S_m}{tp_{up}} = K \cdot \frac{S_s}{tp_{up}} \tag{3-2}$$

其中，tp_{up}为无服务器计算函数与外部存储服务之间上行的吞吐量。

在聚合阶段中，每个聚合节点从外部存储服务中下载其对应的参数分片，在本地执行聚合后，再将聚合后的参数分片回传至外部存储服务中。具体地，每个聚合节点需要获取其余 $W-1$ 个工作节点的参数分片，其中 W 为工作节点的数量。由此，聚合阶段的用时表示如式 3-3所示。

$$\begin{aligned} T_{agg} &= \frac{(W-1)}{K} \cdot \frac{S_m}{tp_{down}} + \frac{1}{K} \cdot \frac{S_m}{tp_{up}} \\ &= (W-1) \cdot \frac{S_s}{tp_{down}} + \frac{S_s}{tp_{up}} \end{aligned} \tag{3-3}$$

其中，tp_{down} 为无服务器计算函数与外部存储服务之间下行的吞吐量。

在下载阶段中，每个工作节点从外部存储服务中下载 K 个聚合后的参数分片，并在本地将各参数分片重组为模型参数。由此，下载阶段的用时表示如式 3-4所示。

$$T_{down} = \frac{S_m}{tp_{down}} = K \cdot \frac{S_s}{tp_{down}} \tag{3-4}$$

在上述建模中，tp_{up} 和 tp_{down} 分别代表无服务器计算函数与外部存储服务之间上行和下行的吞吐量。在第 3.2.1节中，如图 3.2和图 3.3所示，无服务器计算函数与外部存储服务之间的吞吐量与函数的内存配额以及传输的参数分片大小有关。由于外部存储服务通过分布式部署和缓存等优化，通常提供了远高于无服务器计算函数的传输带宽，因此无服务器计算函数与外部存储服务之间的吞吐量的上限受制与函数本身的带宽能力。而无服务器计算函数的网络带宽与其内存配额成正比，因此两者间传输吞吐量的上限与函数的内存配额直接相关。另一方面，到当传输的分片较小时，网络带宽无法被充分利用，此时吞吐量会随着分片大小的减小而降低。

基于上述分析，本文使用一个指数饱和函数（exponential saturation function）来模拟固定内存配额 M 下吞吐量 tp 与分片大小 S_s 之间的关系。具体地，上行吞吐量 tp_{up} 和下行吞吐量 tp_{down} 分别表示如式 3-5和 3-6所示。

$$tp_{up} = p_{up}(M) * \left(1 - e^{-t_{up}(M)*S_s}\right) \tag{3-5}$$

$$tp_{down} = p_{down}(M) * \left(1 - e^{-t_{down}(M)*S_s}\right) \tag{3-6}$$

其中，$p(M)$ 代表在特定内存配额 M 下可达到的最大带宽，而 $t(M)$ 描述了随着分片大小减小，吞吐量的衰减速率。$p(M)$ 和 $t(M)$ 是与选择的无服务器计算平台有关的系数，并且在上传和下载时会有所不同，表示为与内存配额 M 相关的一个函数。4.1节对上述两个系数的测量进行了详细的说明。

基于上述对通信过程的三个阶段的用时分析，可以构建出每轮迭代中聚合节点的通信总用时 T^a_{comm} 如式 3-7所示:

$$\begin{aligned} T^a_{comm} &= T_{up} + T_{agg} + T_{down} \\ &= \frac{S_m}{tp_{up}} + \left(\frac{W-1}{tp_{down}} + \frac{1}{tp_{up}}\right)\frac{S_m}{K} + \frac{S_m}{tp_{down}} \end{aligned} \tag{3-7}$$

为了探究对给定模型大小 S_m、工作节点数量 W 和函数内存配额 M 下通信时间 T_{comm} 与聚合节点数量 K 的关系，式 3-7重新整理后得到式 3-8:

$$T^a_{comm} = \frac{C_1}{K} + C_2 \tag{3-8}$$

其中，C_1 和 C_2 与无服务器计算函数与外部存储服务之间吞吐量有关，具体表示如式 3-9和式 3-10所示。

$$C_1 = S_m\left(\frac{W-1}{tp_{down}} + \frac{1}{tp_{up}}\right) \tag{3-9}$$

$$C_2 = S_m(\frac{1}{tp_{up}} + \frac{1}{tp_{down}}) \tag{3-10}$$

如式 3-1所示，对给定的模型大小 S_m，随着聚合节点数量 K 的减少，参数分片大小 S_s 随之增大。从图 3.2和图 3.3中可知，无服务器计算函数与外部存储服务之间吞吐量 tp_{up} 和 tp_{down} 近似等于无服务器计算函数的网络带宽。此时 $C1$ 和 $C2$ 近似为常量，通信的时间开销 T_{comm} 是聚合节点数量 K 的反比例函数，如式 3-8所示。

当聚合节点数量 K 较大时，模型参数被分割为较小的参数分片。此时，无服务器计算函数与外部存储服务之间的传输吞吐量 tp_{up} 和 tp_{down} 将随之快速衰减。相应地，$C1$ 和 $C2$ 将快速增大，进而导致整体的通信开销 T_{comm} 随之增加。

第 2.2节中，图 2.3和图 2.4展示了通信时间 T_{comm} 随聚合节点数量 K 变化的趋势。对于 ResNet50 模型，其通信时间的变化符合上述的分析。在聚合节点数量 K 小于 7 时，曲线呈现出反比例函数的特征。而当聚合节点数量 K 继续增大时，曲线出现了向上的趋势。最终呈现出一个“U”字形的情况。而对于 SqueezeNet 模型，由于模型本身比较简单，模型参数大小 S_m 较小。因此，此时的参数分片大小 S_s 始终无法充分利用无服务器计算函数的网络带宽能力。因此，随着聚合节点数量 K 的增加，整体的通信时间开销 T_{comm} 不断上涨。整体曲线呈现出 K 从 1 到 2 的轻微下降，以及从 2 之后不断上升的趋势。第 2.2节中的观测实验初步验证了本节中对通信时间开销 T_{comm} 建模的正确性。第 5章实验部分将进一步验证该建模的准确性。

而对于非聚合节点，其通信阶段仅包含上传和下载两个阶段。因此，每轮迭代中非聚合节点的通信总用时 T^n_{comm} 如式 3-11所示。

$$\begin{aligned} T^n_{comm} &= T_{up} + T_{down} \\ &= \frac{S_m}{tp_{up}} + \frac{S_m}{tp_{down}} \end{aligned} \tag{3-11}$$

3.3.1.2　训练时间建模

本节对训练的耗时进行建模。CPU 上计算工作负载的执行时间受到两方面因素的影响，分别是待执行的计算量和可获得的算力资源。在无服务器计算范式下的分布式训练任务中，单次训练迭代的执行时间主要受到批次大小 B（batch size）和无服务器计算函数的内存配额 M 的影响。其中，批次大小 B 决定了单次训练迭代中待训练的数据集的大小，而内存配额 M 则与无服务器计算平台提供给函数的 CPU 算力资源紧密绑定，呈现正相关的关系。

本文测试了 ResNet50 模型和 SqueezeNet 模型在不同内存配额 M 和批次大小 B 下单轮迭代的训练耗时，作为对训练时间开销建模的支撑。图 3.6和图 3.7展示在固定的内存配额 M 下，训练时间与批次大小 B 之间的关系。从图中可以看出，当内存配额 M 不变时，训练时间与批次大小 B 呈现线性关系。由于在无服务器计算平台中函数的 CPU 算力与其内存配额成正比例分配，因此当内存配额 M 固定时，函数的算力资源也随之固定。此时单次迭代中的训练时间随着训练数据量的增加线性增长。

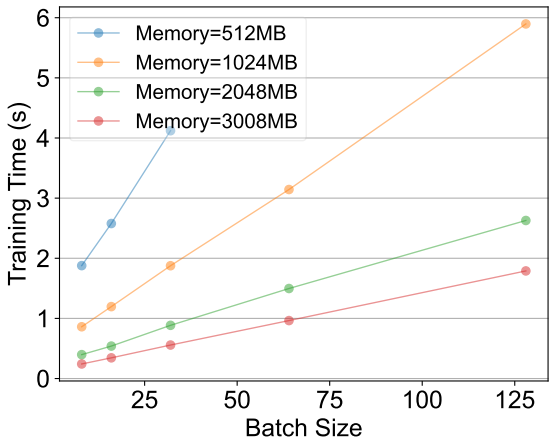


图 3.6　ResNet50 模型训练时间随批次大小的变化情况
Figure 3.6　Training Time of ResNet50 with Different Batch Size

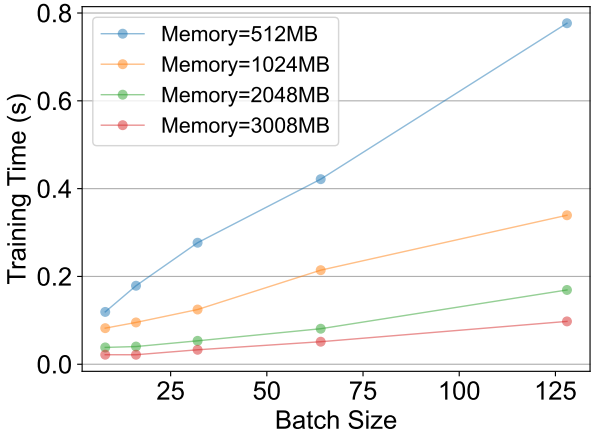


图 3.7　SqueezeNet 模型训练时间随批次大小的变化情况
Figure 3.7　Training Time of SqueezeNet with Different Batch Size

同时，本文在固定的批次大小 B 下测试了训练时间与内存配额 M 之间的关系，

如图 3.8和图 3.9所示。从图中可以看出，当批次大小 B 不变时，训练时间与内存配额 M 呈现反比例关系。即在固定的训练数据量下增加函数的内存配额 M，函数的算力资源也随之增加，训练相同数量的数据集所使用的时间随之下降。

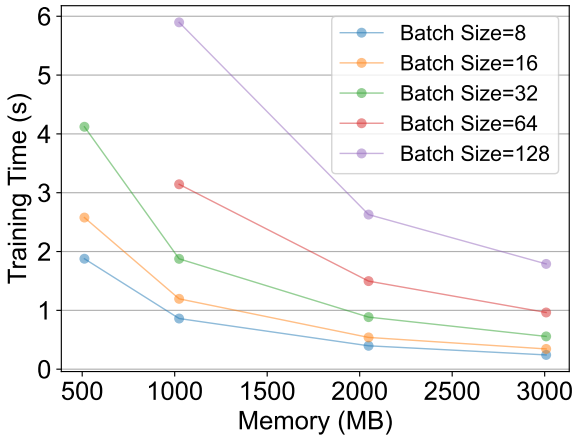


图 3.8　ResNet50 模型训练时间随内存配额的变化情况
Figure 3.8　Training Time of ResNet50 with Different Memory Quota

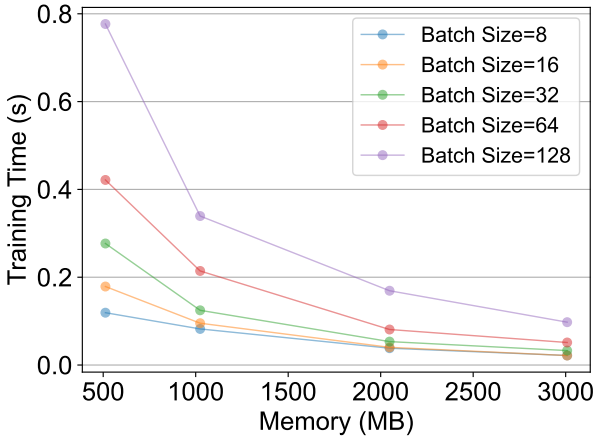


图 3.9　SqueezeNet 模型训练时间随内存配额的变化情况
Figure 3.9　Training Time of SqueezeNet with Different Memory Quota

图 3.6和图 3.8中 ResNet50 模型的一些数据点没有得到测试结果。这是由于随着批次大小 B 的增加，训练模型所需要的最少内存随之增加。这里的内存消耗包括存储完整的训练模型和梯度传播过程中产生的中间数据。因此对于一些内存配额 M 较小而批次大小 B 较大的数据点，无服务计算函数在训练 ResNet50 模型的过程中出现了内存溢出（Out of Memory, OOM）错误。

综上，本节构建了单次迭代的训练时间 T_{train_iter} 与函数内存配额 M 以及训练批次大小 B 之间的关系，如式 3-12所示。整体上看，单次迭代的训练时间 T_{train_iter} 与训练的批次大小 B 成正比关系，与函数的内存配额 M 呈反比关系。

$$T_{train\_iter} = a \cdot \frac{B + b}{M + m} \tag{3-12}$$

其中，a、b 和 m 是与训练模型相关的常量系数，表征了模型训练的复杂程度。第4.1节对上述与训练模型相关的系数的测量进行了详细的说明。

3.3.1.3　初始化时间建模

本节对训练过程中的初始化耗时进行建模。在该阶段中，各工作节点需要从外部存储服务（如 AWS S3）中加载训练模型和对应的分区训练数据到本地，以执行后续的本地训练。由于现有的主流商用无服务器计算平台对无服务器计算函数的最大生命周期进行了限制，例如在 AWS Lambda 平台上，Lambda 函数的最大生命周期为 15 分钟。而一次完整的分布式训练任务由于训练数据集的庞大，通常都远超这个最大时间限制。因此，各个工作节点必须在函数超时前的一个一致的迭代点停止训练，并重新触发新一轮的函数继续执行训练。此外，由于无服务器计算函数的无状态特性（statelessness），每次新触发的函数都需要重新加载模型和数据集。综上，实际训练过程中，每一轮次的无服务器计算函数触发之前都需要执行一次初始化操作。

如图 3.5所示，由于采取了不均等的训练数据划分策略，非聚合节点相比聚合节点将分配到更多的训练数据。因此，整个初始化阶段的耗时实际取决于非聚合节点。第 3.2.2节分析指出，数据划分时分配给不同角色的训练数据量的比例与两者的批次大小 B 的比例保持一致。因此，非聚合节点需要加载的数据总量 S_{load} 表示为式 3-13。

$$S_{load} = S_m + S_d \cdot \frac{B^n}{B^g} \tag{3-13}$$

其中，S_m 为训练模型的大小，S_d 为训练数据集的大小，B^n 为非聚合节点的批次大小，B^g 为全局批次大小。具体地，全局批次大小 B^g 定义为一个迭代轮次中，所有工作节点训练的数据样本总和。因此，B^g 可以表示为如式 3-14所示，即所有工作节点的局部批次大小之和。

$$B^g = B^a \cdot K + B^n \cdot (W - K) \tag{3-14}$$

其中，W 为工作节点的总数，因此聚合节点和非聚合节点的数量分别为 K 和 $W-K$，B^a 为聚合节点的批次大小。

由此，可以构建出初始化阶段加载训练模型和训练数据集的用时 T_{load} 如式 3-13所示。

$$T_{load} = \frac{S_{load}}{tp_{down}} = \frac{S_m + S_d \cdot \frac{B^n}{B^g}}{tp_{down}} \tag{3-15}$$

3.3.1.4　端到端训练时间建模

在对通信阶段、训练阶段和初始化阶段分别进行数学建模后，本节对分布式训练的端到端训练时间进行建模。在本文的设计中，每一个轮次（epoch）的训练是通过触发一轮无服务器计算函数来完成的。每个轮次的时间开销包括了一次性的初始化时间 T_{load} 以及后续每次迭代（iteration）中产生的训练时间 T_{train_iter} 和通信时间 T_{comm}。由此，可以构建出一个轮次的用时 T_{epoch}，如式 3-16所示。

$$\begin{aligned} T_{epoch} &= T_{load} + T_{iter} \cdot I \\ &= T_{load} + (T_{train\_iter} + T_{comm}) \cdot I \end{aligned} \tag{3-16}$$

其中，T_{iter} 代表一次迭代的用时，即迭代中训练时间和通信时间之和，I 代表了一个轮次中的迭代数。由于每个轮次中完成了对训练数据集的一次完整训练。因此，可以求出每个轮次中的迭代数 I，如式 3-17所示。

$$I = \frac{D}{B^g} \tag{3-17}$$

其中，D 为训练数据集中的样本数量，B^g 为全局批次大小，即单次迭代中所有工作节点训练的数据样本的总和。

对于给定的总训练轮次数 E，分布式训练任务的端到端训练时间 $T_{K-REDUCE}$ 可以表示为每个训练轮次的用时 T_{epoch} 和总训练轮次数 E 的乘积，如式 3-18所示。

$$\begin{aligned} T_{K-REDUCE} &= T_{epoch} \cdot E \\ &= (T_{load} + T_{iter} \cdot I) \cdot E \\ &= (T_{load} + (T_{train\_iter} + T_{comm}) \cdot I) \cdot E \end{aligned} \tag{3-18}$$

由于聚合节点和非聚合节点采用了不同的同步策略和数据分配，两者的通信时间 T_{comm} 和训练时间 T_{train_iter} 有所不同，进而体现在每轮迭代的总用时的表达式上的差异。对于非聚合节点，每轮迭代用时 T_{iter}^n 表示为式 3-19。其中，训练时间 $T_{train_iter}^n$的训练数据量为非聚合节点批次大小 B^n，而通信时间 T_{comm}^n 包含了上传和下载两个阶段。

$$\begin{aligned} T_{iter}^n &= T_{train\_iter}^n + T_{comm}^n \\ &= a \cdot \frac{B^n + b}{M + m} + \frac{S_m}{tp_{up}} + \frac{S_m}{tp_{down}} \end{aligned} \tag{3-19}$$

对于聚合节点，其每轮迭代用时 T^a_{iter} 表示为式 3-20。其中，训练时间 $T^a_{train\_iter}$ 的训练数据量为聚合节点批次大小 B^a，而通信时间 T^n_{comm} 包括了上传、聚合和下载三个阶段。

$$\begin{aligned} T^a_{iter} &= T^a_{train\_iter} + T^a_{comm} \\ &= a \cdot \frac{B^a + b}{M + m} + \frac{S_m}{tp_{up}} + \left(\frac{W-1}{tp_{down}} + \frac{1}{tp_{up}} \right) \frac{S_m}{K} + \frac{S_m}{tp_{down}} \end{aligned} \tag{3-20}$$

为了保证聚合节点和非聚合节点之间的同步，应满足两者每次迭代的总用时之间的一致，如式 3-21所示。

$$T^n_{iter} = T^a_{iter} \tag{3-21}$$

将式 3-20和式 3-19代入式 3-21并化简后，可以得到 K-REDUCE 通信模式下聚合节点和非聚合节点在迭代间的同步性约束如式 3-22所示。

$$T^n_{train\_iter} = T^a_{train\_iter} + T_{agg} \tag{3-22}$$

3.3.2　训练开销计费建模

无服务器计算范式下，分布式训练任务采用无服务器计算函数作为训练节点，并借助外部存储服务构建通信通道实现参数聚合。因此，系统的开销计费包括了无服务器计算函数的计费和外部存储服务的计费两个部分。

3.3.2.1　无服务器计算函数计费建模

无服务器计算范式中，平台对用户采用了按使用付费（pay-as-you-go）的计费模式，即仅当无服务器计算函数运行时才对运行过程中占用的资源量进行收费。由于无服务器计算平台采用了以内存为中心的资源配置方案，单个函数的计费与函数配置的内存量 M 成正比。同时，单个函数的计费也与函数的运行时间 T 成正比。假设分布式训练中并行地启动 W 个函数进行训练，训练过程中的函数计费 C_λ 可以表示为式 3-23，其中 p^λ 为单位时间和内存配额下 Lambda 函数的收费。

$$C_\lambda = pr^\lambda \cdot T \cdot M \cdot W \tag{3-23}$$

3.3.2.2　外部存储服务计费建模

通信通道的具体实现可以根据需求选择不同类型的存储服务，如持久对象存储服务（如 AWS S3）或缓存服务（如 AWS ElasticCache）。根据 K-REDUCE 通信模式的特征以及存储服务的计费方式，可以分别构建不同存储服务在训练过程中的计费模型。

对于持久对象存储服务，以 AWS S3 为例，其对存储对象的上传（PUT）和下载（GET）操作计费，而非对对象的存储本身计费。具体地，S3 对上传和下载操作按照其请求次数分别进行收费，且与传输的对象大小无关。根据第 2.2节中对 K-REDUCE 通信模式的分析，可以统计出每轮迭代中的上传请求次数 R_{up} 和下载请求次数 R_{down}。

对于每轮迭代中上传请求次数 R_{up}，在上传阶段中，W 个工作节点将模型参数划分为 K 个分片，然后 $W-K$ 个非聚合节点上传所有 K 个分片， K 个聚合节点上传除自身序号对应分片外的 $K-1$ 个分片。在聚合阶段中，K 个聚合节点上传自身序号对应的聚合完成的分片。因此，每轮迭代中的上传请求次数 R_{up} 如式 3-24所示。

$$\begin{aligned} R_{up} &= (W-K)\cdot K + K\cdot(K-1) + K\cdot 1 \\ &= K\cdot W \end{aligned} \tag{3-24}$$

对于每轮迭代中下载请求次数 R_{down} ，在聚合阶段中，K 个聚合节点下载其余 $W-1$ 个工作节点的分片中与自身序号对应的分片。在下载阶段，$W-K$ 个非聚合阶段下载聚合后的 K 个分片，而 K 个聚合节点下载除自身序号对应的分片外的 $K-1$ 个聚合完成的分片。因此，每轮迭代中的下载请求次数 R_{down} 如式 3-25所示。

$$\begin{aligned} R_{down} &= K\cdot(W-1) + (W-K)\cdot K + K\cdot(K-1) \\ &= 2K\cdot(W-1) \end{aligned} \tag{3-25}$$

综上，整个训练过程中使用 AWS S3 存储服务作为通信通道的总开销计费 C_{S3} 如式 3-26所示，其中 pr^{S3}_{PUT} 和 pr^{S3}_{GET} 为 AWS S3 对一次上传操作和下载操作的收费。

$$C_{S3} = E\cdot I\cdot(pr^{S3}_{PUT}\cdot R_{up} + pr^{S3}_{GET}\cdot R_{down}) \tag{3-26}$$

对于缓存服务，以 AWS ElasticCache 为例，其计费基于传输的对象的大小。在每一轮次的训练中，各节点需要在函数生命周期的开始和结束分别下载和上传一次训练数据集。式 3-24和式 3-25 表示了一轮迭代中所有节点上传和下载的总次数。每次传输的参数分片的大小 S_s 如式 3-1所示。因此，一轮迭代中传输的参数的总大小 S_{iter} 如式 3-27所示。

$$\begin{aligned} S_{iter} &= S_s\cdot(R_{down}+R_{up}) \\ &= S_m\cdot(3W-2) \end{aligned} \tag{3-27}$$

综上，整个训练过程中使用 AWS ElasticCache 缓存作为通信通道的总开销计费 C_{EC} 如式 3-28所示，其中 pr^{EC} 为 AWS ElasticCache 对传输单位大小的对象的收费。

$$C_{EC} = pr^{EC}\cdot E\cdot(2S_d + I\cdot(S_m\cdot(3W-2))) \tag{3-28}$$

3.3.2.3　训练总开销计费建模

整个训练过程中训练任务的总开销计费 $C_{K-REDUCE}$ 等于 Lambda 函数计费和存储服务的计费之和，如式 3-29所示。

$$C_{K-REDUCE} = C_{\lambda} + \begin{cases} C_{S3} & \text{with S3} \\ C_{EC} & \text{with ElasticCache} \end{cases} \tag{3-29}$$

3.3.3　模型收敛性能约束

上文对训练任务的端到端训练时间进行了建模，而训练任务的效果还与模型的收敛速度有关。模型的训练损失（training loss）同时受到训练的轮次和全局的批次大小这两个因素的影响。研究[13]中指出，在用户指定了总体的训练轮次后，随着全局批次大小的增加，模型通常需要更多的训练轮次才能达到指定的训练损失值。也就是说，模型的收敛速度与全局批次大小成负相关关系。因此在训练轮次固定的情况下，需要限制全局批次大小以确保模型的收敛效率。式 3-14表示了训练的全局批次大小。针对不同模型的特点，本文设置一个最大全局批次大小 B^{g}_{max}，作为对模型收敛性能的约束，如式 3-30所示。

$$B^{g} = B^{a} \cdot K + B^{n} \cdot (W - K) \leq B^{g}_{max} \tag{3-30}$$

3.4　自动化参数配置模块

在无服务器计算范式下，分布式训练的性能与多个参数的配置相关，且多个参数与模型不同性能之间存在高度耦合的关系。如第 2.3节中分析，错误的参数配置会导致模型的性能的衰退。因此，开发者在无服务器计算范式下部署分布式训练任务时，面临着参数配置困难的挑战。为此，本节探究参数配置的优化方案，以提高模型训练的整体性能表现，并降低用户的配置负担。首先，3.4.1节构建了无服务器计算范式下分布式训练的性能优化问题，并梳理涉及的关键配置参数。其次，3.4.2节采用剪枝策略缩小参数配置的搜索空间，以降低问题的复杂度。最后，3.4.3节提出了一个适用于本训练架构的两阶段启发式搜索算法，以实现高效的自动化参数配置。

3.4.1　性能优化问题

本节构建了无服务器计算范式下分布式训练的性能优化问题。第 3.3节性能建模模块构建了训练的三个性能指标，分别是端到端训练时间 $T_{K-REDUCE}$、训练开销计费 $C_{K-REDUCE}$ 以及模型的收敛效率。根据式 3-18、式 3-29和式 3-30可知，在 K-

REDUCE 通信模式下，模型的训练性能与以下五个参数有关，分别是函数的内存配额 M、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n。

分布式训练任务的效率由训练任务的端到端训练时间和收敛效率两个因素保证。因此，在参数配置中，需要对最大的训练时间 T_{max} 和最大的训练批次 B^g_{max} 进行约束和限制。性能优化问题的目标是在上述性能约束得到保证的前提下进行参数搜索，以最小化训练的开销计费。同时，K-REDUCE 通信模式需要保证聚合节点和非聚合节点之间的同步性约束，如式 3-22所示。综上，K-REDUCE 通信模式下分布式训练任务的参数配置优化问题可以建模为式 3-31所示。

$$\begin{aligned} \min_{M,W,K,B^a,B^n} \quad & C_{K-REDUCE} \\ \text{s.t.} \quad & T \le T_{\max}, \\ & B^g \le B^g_{\max}, \\ & T^n_{train\_iter} = T^a_{train\_iter} + T_{agg} \end{aligned} \tag{3-31}$$

将式 3-8、式 3-11、式 3-12、式 3-15和式 3-17代入式 3-18，并将式 3-23、式 3-26和式 3-28后可知，端到端的训练时间 $T_{K-REDUCE}$ 和训练的开销计费 $C_{K-REDUCE}$ 与函数的内存配额 M、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n 呈非线性（non-linear）关系。相应地，将端到端的训练时间 $T_{K-REDUCE}$ 和训练的开销计费 $C_{K-REDUCE}$ 的表达式代入到式 3-31中，上述优化问题变成了一个非线性整数规划（non-linear integer programming）问题。这类问题的求解是 NP 困难（NP-hard）[60]，因此，本节设计了一个参数剪枝策略和一个两阶段启发式搜索算法，以降低参数配置优化问题的复杂度，并提升系统架构的自动化参数配置性能。

3.4.2　参数剪枝策略

为了解决上述 NP 困难的优化问题，本节对五个核心参数的搜索空间进行剪枝，以有效地搜索最优参数。下文依次对函数的内存配额 M、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n 五个参数的约束条件进行分析，并给出各个参数的搜索范围的上界与下界。

3.4.2.1　聚合节点和非聚合节点的批次大小 B^a 和 B^n

模型训练中，批次大小（batch size）代表了一轮迭代中训练的数据样本数量。本文定义一轮迭代中单位时间内训练的数据样本数量为训练速率（training rate）。根据

式 3-12构建的训练时间，训练节点的训练速率 TR^a 如式 3-32所示。

$$TR = \frac{B}{T_{train\_iter}} = \frac{M+m}{a} \cdot \frac{1}{1+\frac{b}{B}} \tag{3-32}$$

式中，a、b 和 m是与训练模型相关的常量系数。因此，对于给定的训练模型，其训练速率的上限为 $\frac{M+m}{a}$，与函数的内存配额相关。而训练节点的批次大小 B 则代表了实际的训练速率占最大训练速率的比例 γ，如式 3-33所示。

$$\gamma = \frac{1}{1+\frac{b}{B}} \tag{3-33}$$

随着训练节点批次大小 B 的增加，模型的训练速率随之增加。为了获得较高的训练速率，以加快训练的速度，本文设置了一个最小的比例 γ_{min}，作为批次大小的下限值。随着批次大小的增加，模型训练所需的内存也随之增加。训练对应批次大小所需的最小可执行内存不应超过函数的内存配额。因此，可以根据函数的内存配额确定训练批次的上限值。综上，训练节点的批次大小的搜索范围如式 3-34所示。其中，B_M 为内存配额 M 下可执行训练的最大批次大小。

$$\frac{b}{\frac{1}{\gamma_{min}}-1} = B_{lower} \le B \le B_{upper} = B_M \tag{3-34}$$

由于 K-REDUCE 通信模式中聚合节点和非聚合节点的批次大小不同，并且优化问题式 3-31中综合考虑了最大全局批次大小与两类节点间的同步性约束。因此，在配置批次大小时，聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n 还应满足上述两个约束条件，如式 3-35所示。

$$\begin{gathered} B^a \cdot K + B^n \cdot (W-K) \le B^g_{max} \\ a \cdot \frac{B^n+b}{M+m} = a \cdot \frac{B^a+b}{M+m} + (W-1) \cdot \frac{S_s}{tp_{down}} + \frac{S_s}{tp_{up}} \end{gathered} \tag{3-35}$$

由于训练节点的批次大小的搜索范围与工作节点数量 W 和内存配额 M 相关，因此参数搜索中应先确定工作节点数量 W 和内存配额 M，再确定聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n。此外，训练批次的大小通常设为 16、32 或更大的批次大小。为了在参数搜索的效率与搜索结果的精度之间取得平衡，本文在搜索最优批次大小时设置搜索步长为 16。

3.4.2.2　工作节点的数量 W

在确定了训练批次的下限 B_{lower} 后，工作节点数量 W 的上限受制于最大的全局批次大小 B^g_{max}。由此，工作节点数量的搜索范围如式 3-36所示。

$$1 = W_{lower} \le W \le W_{upper} = \frac{B^g_{max}}{B_{lower}} \tag{3-36}$$

工作节点数量 W 的搜索范围从下限值 1 至上限值 W_{upper}。当工作节点的数量为 1 时，分布式训练退化为单机训练（stand-alone training）。第 5.6节中对分布式训练和单机训练的性能进行了对比。

3.4.2.3　函数的内存配额 M

对于函数的内存配额 M, 无服务器计算平台限制了用户可设置的配额区间。例如，AWS Lambda 平台设置了单个函数的内存配额区间从 1 MB 到 10240 MB，且配置粒度为 1 MB。本节从训练任务运行要求和配置搜索效率两方面，对函数的内存配额 M 的选择进行分析。

对于给定的训练模型和批次大小，训练所需的内存需要达到指定的最小可运行内存。具体地，在神经网络中，每一层都有与之相关的权重（weights）和偏置（biases）。这些参数在训练过程中需要存储在内存中，并且随着模型的规模增大（层数增多、每层神经元数量增加），权重和偏置占用的内存空间会显著增加。同时，在训练过程中需要计算损失函数关于模型参数的梯度以更新模型参数。对于每一个可训练的参数，都需要存储其梯度。例如，在反向传播算法中，通过计算梯度来确定如何调整权重和偏置。这些梯度的大小与模型参数的大小相同。所以如果模型有大量的参数，梯度存储也会消耗一定的内存。此外，在模型的前向传播过程中，会产生许多中间计算结果。这些结果需要存储在内存中，直到后续层的计算完成。而且，在反向传播过程中也需要这些中间结果来计算梯度。训练数据是以批次（batch）为单位输入到模型中的，因此，每一批次的数据都需要存储在内存中。第 4.1节中构建一个负载分析模块（*Profiler*），对模型训练所需的最小可运行内存进行详细的分析和建模。

在训练过程中，函数的内存配额需要满足上述所有需求，因此需要达到指定的最小内存配额。该配额即为内存 M 的搜索范围的下限。而内存的最大配额不能超过无服务器计算函数平台可分配的最大内存。由于在无服务器计算平台上，函数的内存和函数的算力相关，因此，函数的内存配额除了需要满足可运行要求外，也会影响到训练效率。在最优参数配置的搜索中，尽管无服务器计算平台提供了 1 MB 的分配精度，但为了取得分配精度与搜索效率之间的平衡，本文选择 128 MB 作为搜索步长。该搜索步长选择通过轻微的精度损失，极大地加速了搜索效率。

3.4.2.4　聚合节点的数量 K

K-REDUCE 通信模式将工作节点进行了分类，因此引入了一个额外的参数，即聚合节点的数量 K。在 W 个工作节点中，K 个节点作为聚合节点，其余 $W-K$ 个节

点作为非聚合节点。因此，聚合节点 K 的数量的取值范围从 1 到 W。当聚合节点的数量为 1 时，K-REDUCE 通信模式特例化为 AllReduce 通信模式，即选择单一的节点（leader）作为聚合节点。而当聚合节点的数量为 W 时，K-REDUCE 通信模式特例化为 ScatterReduce 通信模式，即所有节点都作为聚合节点参与聚合阶段的参数聚合。

K-REDUCE 通信模式的优化来源于两个方面。一方面，通过选择恰当的聚合节点数量 K，以最小化通信阶段的开销。另一方面，通过充分利用聚合阶段中非聚合节点的 CPU 算力资源，进一步加速训练过程。因此，最优的聚合节点数量 K 的取值，即为上述两个优化的总和取得最大值的点。

3.4.3　两阶段启发式搜索算法

本节基于上一节的参数剪枝结果，求解式 3-31所建的优化问题。如第 3.4.1节中分析，该优化问题是一个非线性整数规划（non-linear integer programing）问题，其求解是 NP 困难（NP-hard）的。因此，本节提出了一个适用于本训练架构的两阶段启发式搜索算法。

在 K-REDUCE 通信模式下，一共有五个参数需要配置，分别是函数的内存配额 M、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n。其中，由于 K-REDUCE 通信模式中引入了执行异步训练的非聚合节点，因此额外引入了两个参数，分别是聚合节点的数量 K 和非聚合的批次大小 B^n。而在 ScatterReduce 通信模式中，共涉及了 3 个参数，分别是内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a。为了简化 K-REDUCE 通信模式下的参数搜索问题，本节首先将其与 ScatterReduce 通信模式进行对比和分析。引理 3.1和引理 3.2证明，在相同的内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a 配置下，通过选择最优的聚合节点数量 K 和恰当的非聚合节点批次大小 B^n，K-REDUCE 通信模式下的端到端训练时间和训练开销不超过 ScatterReduce 通信模式下的端到端训练时间和训练开销。

引理 3.1　$\min\limits_{K^*,B^{n*}} T_{K-REDUCE} \le T_{ScatterReduce}$

证明　在相同的内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a 配置下，证明在最优的聚合节点数量 K 和满足同步性约束的非聚合节点批次大小 B^n 下，K-REDUCE 通信模式的端到端训练时间小于等于 ScatterReduce 通信模式的端到端训练时间。

第 3.3.1.4节指出在满足同步性约束条件式 3-21时，K-REDUCE 的端到端训练时间可以根据聚合节点的用时进行表示。结合第 2.2节的观察和式 3-7对聚合节点通信时间的建模可知，通过选择最优的聚合节点数量 K 可以在聚合阶段的并行度和函数网络带宽之间取得最优平衡，从而最小化通信阶段的耗时。因此，可以得出下述结论。

$$\min_{K^*,B^{n*}} T_{comm} \leq T_{comm}(K = W, B^n = B^a) \tag{3-37}$$

另外，根据式 3-12可知，训练阶段的用时与函数的内存配额 M 以及训练的批次大小 B 有关。因此，K-REDUCE 通信模式下聚合节点的训练时间与 ScatterReduce 通信模式相同。从而一轮迭代的总时长，K-REDUCE 通信模式亦优于 ScatterReduce 通信模式。

$$\min_{K^*,B^{n*}} T_{train\_iter} \leq T_{train\_iter}(K = W, B^n = B^a) \tag{3-38}$$

在 K-REDUCE 通信模式中，选择恰当的非聚合节点批次大小 B^n 需要保证式 3-21的同步性约束得到保证，即非聚合节点在一轮迭代中分配到的批次大小 B^n 大于 B^a。因此，K-REDUCE 通信模式在相同的工作节点的数量 W 和聚合节点的批次大小 B^a 下的全局批次大小 B^g 亦大于 ScatterReduce 通信模式的全局批次大小。给定相同的数据样本数量 D，ScatterReduce 通信模式在每个训练轮次中需要经过更多轮迭代。

$$\min_{K^*,B^{n*}} I \leq I(K = W, B^n = B^a) \tag{3-39}$$

而对于每一轮迭代初的加载用时，ScatterReduce 通信模式略优于 K-REDUCE 通信模式。但该一次性的优势对比上述多轮迭代的增益较小。因此，根据式 3-18构建的端到端训练时间可知，在最优的聚合节点数量 K 和满足同步性约束的非聚合节点批次大小 B^n 下，K-REDUCE 通信模式的端到端训练时间小于等于 ScatterReduce。由此，可以得出引理 3.1。 ■

引理 3.2 $\min_{K^*,B^{n*}} C_{K-REDUCE} \leq C_{ScatterReduce}$

证明　在相同的内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a 配置下，证明在最优的聚合节点数量 K 和满足同步性约束的非聚合节点批次大小 B^n 下，K-REDUCE 通信模式的训练开销小于等于 ScatterReduce 通信模式的训练开销。

引理 3.1证明了在上述条件下，K-REDUCE 通信模式的端到端训练时间小于等于 ScatterReduce 通信模式的端到端训练时间。根据函数计费模型式 3-23可知，在相同

的内存配额 M和工作节点的数量 W 下，其函数计费亦小于等于 ScatterReduce 通信模式的函数计费。

$$\min_{K^*,B^{n*}} C_\lambda \leq C_\lambda(K=W, B^n=B^a) \tag{3-40}$$

而 ScatterReduce 通信模式中使用的聚合节点的数量更多，进而使得模型参数分片更多。根据式 3-25和 3-24可知，如果使用 S3 作为通信通道，其计费在 ScatterReduce 通信模式下更高。而若使用 ElasticCache 作为通信通道，按照数据传输量进行计费，则两者的开销相同。

$$\begin{aligned}\min_{K^*,B^{n*}} C_{S3} &\leq C_{S3}(K=W, B^n=B^a)\\ \min_{K^*,B^{n*}} C_{EC} &= C_{EC}(K=W, B^n=B^a)\end{aligned} \tag{3-41}$$

综上，根据式 3-29，K-REDUCE 通信模式下训练的总开销计费小于等于 ScatterReduce 通信模式。由此，可以得出引理 3.2。

■

第 5.4节对 K-REDUCE 通信模式的优化性能进行了实验观测。为了验证上述引理的正确性，该实验为各组负载设置了相同内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a 配置下 ScatterReduce 通信模式的对照组（counterpart），以对比两者的端到端训练时间和训练开销计费。

基于上述洞察，可以将 K-REDUCE 通信模式的五个参数划分为两组：（1）内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a；（2）聚合节点的数量 K 和非聚合节点的批次大小 B^n。第一组的三个参数是与 ScatterReduce 通信模式相关的通用参数，而第二组的两个参数则是 K-REDUCE 通信模式额外引入的参数。据此，本节提出了一个适用于本训练架构的两阶段启发式搜索算法，将参数搜索的过程划分为两个独立的阶段，分别求解上述两组参数。

3.4.3.1　收缩因子 δ 下最小化 ScatterReduce 训练开销

本阶段对内存配额 M、工作节点的数量 W 和聚合节点的批次大小 B^a 三个参数进行搜索。引理 3.1和引理 3.2证明，在相同的参数配置下，ScatterReduce 通信模式是 K-REDUCE 通信模式性能的上界。因此，本阶段将优化问题式 3-31的目标重构为求取 ScatterReduce 通信模式下的最优参数配置，以此逼近 K-REDUCE 的最优参数配

置。重构后的目标优化问题如式 3-42所示。

$$\begin{aligned} \min_{M,W,B^a} \quad & C_{ScatterReduce} \\ \text{s.t.} \quad & T \le \frac{T_{max}}{\delta} \\ & B^g \le B^g_{max} \cdot \delta \end{aligned} \tag{3-42}$$

引理 3.1和引理 3.2证明，在相同的配置参数下，K-REDUCE 通信模式下的端到端训练时间相比 ScatterReduce 通信模式更短。同时，K-REDUCE 通信模式下的全局批次大小由于非聚合节点的批次大小的增大而变得更大。因此，在构建本阶段的目标优化问题时，本文设置了一个收缩因子 δ（$0 < \delta \le 1$），用于放宽求解 ScatterReduce 通信模式下配置的端到端时间约束，同时适当收缩对全局批次大小的约束。

本阶段以 ScatterReduce 通信模式下的训练性能作为优化目标。在 ScatterReduce 通信模式下，所有工作节点都作为聚合节点参与聚合，因此可以看作是 K-REDUCE 通信模式下，聚合节点数量 K 等于工作节点数量 W，非聚合节点批次大小 B^n 等于聚合节点批次大小 B^a 的特例化。将 $K = W$ 和 $B^n = B^a$ 代入到式 3-14、式 3-18和式 3-29中，可以得到 ScatterReduce 通信模式下的全局批次大小、端到端训练时间和训练开销。

本阶段参数搜索算法的伪代码如算法 3.1所示。搜索过程按照工作节点的数量 W、内存配额 M 和聚合节点的批次大小 B^a 的顺序依次嵌套遍历。各个参数的搜索空间根据第 3.4.2节中的剪枝结果确定。具体地，首先根据批次大小下限值 B_{lower} 确定工作节点数量的上限值 W_{upper}。然后从 1 至上限值 W_{upper} 按步长 1 进行遍历。对内存配额 M 的搜索从上限值 M_{upper} 向下按步长 128 MB 进行遍历。对批次大小 B^a 搜索从其下限值 B^a_{lower} 开始按步长 16 进行遍历，直到遍历至当前内存配额下可训练的最大批次大小或全局批次大小超过了指定的最大全局批次，作为当前内存配额 M 和工作节点数量 W 下对应的批次大小的上限值 B_{upper}。其中，在遍历内存配额 M 的过程中，若特定内存配额下没有参数配置能满足时间约束，那么进一步缩小内存配额也无法满足该时间约束。因此，本文在此时采用早停策略，直接进入到下一个工作节点数量的迭代中，从而避免不必要的参数搜索，以加速参数搜索的过程。在满足训练时间约束和全局批次大小约束下，算法 3.1搜索出 ScatterReduce 通信模式下的最小训练开销 $C_{ScatterReduce}$ 以及对应的工作节点的数量 W、内存配额 M 和聚合节点的批次大小 B^a。

算法 3.1 收缩因子 δ 下最小化 ScatterReduce 训练开销的参数搜索算法

Input: 深度训练模型: $model$; 训练数据集: $dataset$; 训练总轮次: E; 最大时间约束: T_{max}; 最大全局批次大小: B^g_{max}; 收缩因子: δ.

Output: 聚合节点批次大小: B^{a*}; 工作节点数量: W^*; 内存配额: M^*.

Initialize: $C_{ScatterReduce} \leftarrow \infty$;
$B_{lower} \leftarrow$ Eq 3-34;
$W_{upper} \leftarrow$ Eq 3-36;
foreach W **in** $[1, W_{upper}]$ **do**
    $K \leftarrow W$;
    foreach M **in** $[M_{upper}, 0]$ **with step** -128 **do**
        $B_{upper} \leftarrow Profiler$;
        $early\_stop \leftarrow$ false;
        foreach B^a **in** $[B_{lower}, B_{upper}]$ **with step** 16 **do**
            $B^n \leftarrow B^a$;
            $B^g \leftarrow$ Eq 3-14;
            if $B^g > B^g_{max} \cdot \delta$ **then**
                break;
            $T \leftarrow$ Eq 3-18;
            if $T > \frac{T_{max}}{\delta}$ **then**
                $early\_stop \leftarrow$ true;
                break;
            $C \leftarrow$ Eq 3-29;
            if $C < C_{ScatterReduce}$ **then**
                $W^* \leftarrow W, M^* \leftarrow M, B^{a*} \leftarrow B^a, C_{ScatterReduce} \leftarrow C$;
        if $early\_stop$ **then**
            break;
return W^*, M^*, B^{a*};

3.4.3.2　训练时间约束和收敛速率约束下最小化 K-REDUCE 训练开销

本阶段根据上一阶段确定的工作节点的数量 W、内存配额 M 和聚合节点的批次大小 B^a，以式 3-31作为目标优化问题进一步确定 K-REDUCE 通信模式额外引入的聚合节点数量 K 和非聚合节点的批次大小 B^n。

本阶段参数搜索算法的伪代码如算法 3.2所示。首先，基于算法 3.1确定工作节点的数量 W、内存配额 M 和聚合节点的批次大小 B^a 三个参数。然后，从 1 至 W 依次遍历聚合节点数量 K, 同时基于同步性约束式 3-22确定对应 K 值下的非聚合节点的批次大小 B^n，最终返回在满足训练时间和收敛速率约束条件下最小化 K-REDUCE 训练开销的参数配置。由于在第一阶段的参数搜索中引入了一个收缩因子 δ 作为对

算法 3.2 训练时间约束和收敛速率约束下最小化 K-REDUCE 训练开销的参数搜索算法

Input: 深度训练模型: $model$; 训练数据集: $dataset$; 训练总轮次: E; 最大时间约束: T_{max}; 最大全局批次大小: B^g_{max}.

Output: 聚合节点批次大小: B^{a*}; 工作节点数量: W^*; 内存配额: M^*; 非聚合节点批次大小: B^{n*}; 聚合节点数量: K^*; 最小训练开销: $C_{K-REDUCE}$.

Initialize: $C_{K-REDUCE} \leftarrow \infty$;
foreach δ **in** {0.6, 0.7, 0.8, 0.9, 1.0} **do**
    $W, M, B^a \leftarrow$ Algo 3.1;
    foreach K **in** $[1, W]$ **do**
        $B^n \leftarrow$ Eq 3-22;
        $B^g \leftarrow$ Eq 3-14;
        $T \leftarrow$ Eq 3-18;
        if $B^g > B^g_{max}$ **or** $T > T_{max}$ **then**
            continue;
        $C \leftarrow$ Eq 3-29;
        if $C < C_{K-REDUCE}$ **then**
            $W^* \leftarrow W, M^* \leftarrow M, B^{a*} \leftarrow B^a, K^* \leftarrow K, B^{n*} \leftarrow B^n, C_{K-REDUCE} \leftarrow C$;
return $W^*, M^*, B^{a*}, K^*, B^{n*}, C_{K-REDUCE}$;

ScatterReduce 通信模式和 K-REDUCE 通信模式最优性能之间偏差比例的预测，因此，本阶段以不同的收缩因子 δ 重复上述搜索过程，并返回多轮搜索中的最优结果。

第 2.2节中观察到对于不同类型的训练模型，其最优的聚合节点数量 K 的选择偏好不同。对于结构较为简单、参数规模较小的模型如 SqueezeNet，其倾向于较少的聚合节点数量 K 以减少过度分片导致的通信吞吐衰减的影响。因此，该类模型在本阶段中有更大的优化空间，偏向于选择更小的收缩因子 δ。相反，对于结构较为复杂、参数规模较大的模型如 ResNet50，其最优参数配置偏向于出现在较大的收缩因子 δ 下。

本阶段确定的聚合节点数量 K 和非聚合节点的批次大小 B^n 代表了 K-REDUCE 通信模式下的两个优化效果。其中，最优的聚合节点数量 K 在聚合操作的并行度和网络带宽之间取得最优平衡，以加速参数聚合的通信过程，而非聚合节点的批次大小 B^n 则重复利用了非聚合节点在聚合阶段的空闲 CPU 算力进一步加速了训练过程。因此，本阶段确定的两个参数综合了上述两方面的优化，以在 ScatterReduce 通信模式上进一步加速端到端的训练时间，并减少整体的训练开销。

3.4.3.3 搜索算法时间复杂度分析

本工作提出的两阶段启发式搜索算法将参数搜索过程划分为两个串行的步骤。算法 3.1通过对工作节点的数量 W、内存配额 M 和聚合节点的批次大小 B^a 三个参数的嵌套遍历，确定了满足对应收缩因子 δ 下最小化 ScatterReduce 通信模式训练开销的参数配置。因此，算法 3.1的时间复杂度为 $O(p \cdot q \cdot l)$。其中，$p = W_{upper} - W_{lower} + 1$ 代表了工作节点数量 W 的搜索范围，$q = \frac{M_{upper} - M_{lower}}{M_{step}} + 1$ 代表了内存配额 M 搜索空间的大小，$l = \frac{B_{upper} - B_{lower}}{B_{step}} + 1$ 代表了节点的批次大小 B 搜索空间的大小。而算法 3.2从 1 到 W 遍历聚合节点数量 K，并通过同步性约束式 3-22确定非聚合节点的批次大小 B^n。因此，算法 3.2的时间复杂度为 $O(W)$，其中，W 为算法 3.1确定的工作节点的数量。

在固定收缩因子 δ 下的一次遍历中，两阶段的搜索过程是串行的。因此，两阶段搜索的时间复杂度为 $O(p \cdot q \cdot l + W)$。另外，第二阶段的参数搜索以不同的收缩因子 δ 重复了上述的搜索过程。但由于收缩因子 δ 的取值空间的大小是常量级的，因此并不影响整体算法的时间复杂度。而采用暴力搜索算法时，需要对每个参数进行嵌套遍历，其时间复杂度为 $O(p \cdot q \cdot l \cdot W)$。本文提出的两阶段启发式搜索算法将搜索过程划分为两个独立的阶段，使得参数搜索的时间复杂度从 $O(p \cdot q \cdot l \cdot W)$ 降低至 $O(p \cdot q \cdot l + W)$。另外，参数搜索空间的剪枝检测和搜索过程中的早停策略进一步减少了参数的搜索空间，从而提高搜索算法的性能。总的来说，该两阶段启发式搜索算法为优化问题式 3-31高效地获取了一个接近最优解的结果，以较少的精度损失换取较高的搜索效率。

3.5 本章小结

本章对提出的无服务器计算范式下的高性能分布式训练架构 FasDL 的设计进行了详细介绍。首先，对该系统的整体架构以及工作流程进行概述。然后，分别对该架构中的三个核心模块进行设计，分别是 K-REDUCE 通信模式、性能建模模块以及参数配置优化模块。K-REDUCE 通信模式有效降低了无服务器计算范式下，函数间通信高开销的问题。性能建模模块和参数配置优化模块实现了高效的自动化参数配置和参数调优。因此，该架构有效解决了当前无服务器计算范式下分布式训练任务面临的高通信开销和参数配置困难问题。

第 4 章 无服务器计算范式下分布式训练架构实现

基于第 3章中 FasDL 训练框架的设计，本文在 AWS Lambda 平台上实现了一个 FasDL 的原型。在通信通道的实现上，本文选择了对象存储服务 AWS S3 和缓存服务 AWS ElasticCache 两个存储服务。本章对负载分析模块以及基于 K-REDUCE 通信模式的训练框架的实现细节进行说明。负载分析模块在训练开始前分别对模型负载以及无服务器计算平台的特征进行分析，作为参数搜索过程中对训练性能预测的支撑。第 4.1节对负载分析模块的实现进行说明。而在 K-REDUCE 通信模式下，分布式训练由无服务器计算函数担任的不同类型的工作节点和基于外部存储实现的通信通道协同交互完成。第 4.2节对两者在训练架构中的实现进行说明。

4.1 负载分析模块实现

本节构建了一个负载分析器 *Profiler*，分别对训练模型和无服务器计算平台的特征进行分析。对于训练模型，负载分析器 *Profiler* 确定其训练所需的最小可运行内存和训练性能相关系数 a、b 和 m。对于无服务器计算平台，负载分析器 *Profiler* 确定 Lambda 函数的通信相关的系数 $t(M)$ 和 $p(M)$。

4.1.1 模型训练性能分析

模型训练性能的分析包括了模型训练相关系数拟合和最小可运行内存建模两个部分，以支持训练性能的建模和参数搜索的剪枝。第 3.3节构建了模型训练时间 T_{train_iter} 与训练批次大小 B 和函数内存配额 M 之间的关系，如式 3-12所示。其中，与训练模型相关的系数包括 a、b 和 m。而第 3.4节在对节点训练批次 B 的搜索范围进行剪枝时，通过特定批次大小 B 下训练所需的最小可运行内存确定其上限值。

4.1.1.1 模型训练相关系数

如式 3-12所示，与模型训练性能相关的系数包括 a、b 和 m 三个。模型的训练时间 T_{train_iter} 与训练批次大小 B 和函数内存配额 M 相关。函数的内存配额 M 与函数的算力相关，而训练批次大小 B 决定了一轮批次内训练数据的数量。因此，负载分析器 *Profiler* 通过在不同训练批次大小 B 和函数的内存配额 M 的组合下多次触发对应训练模型的训练，并记录对应一轮迭代的训练时间。接下来，基于式 3-12对一轮迭代中训练时间的建模，采用最小二乘法（Least - Squares Method）对测试数据进

行拟合，求解出各个训练模型的训练相关系数。

由于与训练相关的系数只与训练模型本身有关，因此在该分析过程中不涉及无服务器计算函数的通信。在固定的内存配额 M 下训练时间与训练批次大小 B 成线性关系，而在固定的训练批次大小 B 下训练时间与内存配额 M 成反比例关系。系数 b 和 m 分别表征了训练批次大小 B 和内存配额 M 两者与训练时间在上述关系中的偏移量，而系数 a 则表征了在特定函数算力下训练一定数量数据所需的时间量级。因此，对于结构较为复杂、模型参数较多的模型，其系数 a 一般也越大。第 5.1节实验设置中给出了对实验负载进行了上述分析和拟合的结果，其对应的训练系数如表 5.3所示。与分析一致，系数 a 与模型的复杂度呈现正相关特点，而系数 b 和 m 则与模型本身的特性相关。

4.1.1.2　最小可运行内存

参数搜索中需要保证在训练批次大小 B 与函数内存配额 M 之间的约束关系，即函数内存配额 M 大于训练对应批次大小 B 所需的最小可运行内存。在模型训练过程中，内存的使用主要包括三个部分，分别是训练模型、训练批次以及训练中间结果。训练模型的大小 S_m 是固定的，且训练模型需要在训练期间常驻内存。训练数据是以批次为单位输入到模型中的，因此每一批次的数据都需要存储在内存中。训练批次的大小由训练的样本大小和训练的批次大小 B 决定。另外，每一轮迭代包括了前向传播和反向传播两个过程。在反向传播的过程中，需要使用前向传播产生的中间结果来计算梯度。训练产生的中间结果的大小与模型本身的结构以及训练的批次大小 B 有关。因此，除训练模型的内存消耗外，其余部分都与批次大小 B 的选择有关，且成正相关关系。因此，批次大小 B 下的最小可运行内存 M_B 如式 4-1所示。

$$M_B = k \cdot B + c \tag{4-1}$$

其中，系数 k 与模型的训练特性和训练数据有关，表征了训练单位数据样本所消耗的内存量。而系数 c 则包含了前者的常量系数以及模型本身消耗的内存量。

在上述模型训练相关系数 a、b 和 m 的分析过程中，同时记录对应批次大小 B 下实际消耗的内存使用量 M_B，并通过最小二乘法（Least - Squares Method）进行线性拟合，可以确定系数 k 和 c。由此，也可以确定在对应的内存配额 M 下，训练批次大小 B 的上限值 B_M 如式 4-2所示。

$$B_M = \frac{M - c}{k} \tag{4-2}$$

4.1.2 无服务器计算平台通信性能分析

在对无服务器计算平台的通信性能进行分析时，需要确定函数在数据传输过程中的上行和下行的吞吐能力。第 3.2.1节中观察到函数的吞吐量与函数的内存配额 M 以及传输的分片大小 S_s 有关。第 3.3.1节中用指数饱和函数来模拟在固定内存配额 M 下，吞吐量 tp 与分片大小 S_S 之间的关系。式 3-5和 3-6中，$p(M)$ 和 $t(M)$ 分别是在特定内存配额 M 下的最大带宽和吞吐量的衰减速率。在不同的内存配额 M 下，函数的吞吐量 tp 随传输的分片大小 S_S 的变化情况有所不同。因此，$p(M)$ 和 $t(M)$ 实际上是不同内存配额 M 下的两组系数集合。

负载分析器 *Profiler* 通过触发不同内存配额 M 的函数与外部存储进行数据传输。通过记录传输不同分片大小 S_S 的上行和下行时间并计算对应的传输吞吐量，可以得到在对应内存配额 M 下吞吐量 tp 随分片大小 S_S 的变化情况。然后，采用最小二乘法（Least - Squares Method）进行拟合，可以确定对应的系数 $p(M)$ 和 $t(M)$。第 3.4.2节中选择 128 MB 作为内存配额 M 的遍历步长。因此，本节同样采用 128 MB 作为上述分析中内存配额 M 的遍历间隔，依次得到多组系数。另外，由于内存配额 M 增大到对应的临界值时，函数的吞吐量趋于一个最大值而不再增加。因此，可以不再对后续内存配额 M 进行测试，从而减少不必要的系统开销。第 5.1节中展示了 AWS Lambda 与 S3 之间传输吞吐量的拟合结果。由于$p(M)$ 和 $t(M)$ 仅与无服务器计算平台本身提供的性能有关，而与训练负载无关。因此，对于不同的训练模型，只需执行一次上述分析过程。

4.2 基于 K-REDUCE 通信模式的训练框架实现

图 3.5详细描述了 K-REDUCE 通信模式下的训练流程，包括聚合节点数量 K 的选择、训练数据集的划分以及 K-REDUCE 通信模式下的分布式训练。本节对 K-REDUCE 训练框架的一些实现细节进行说明。首先，第 4.2.1节对训练初始化阶段的实现进行说明，包括训练参数配置、训练数据集的划分以及训练工作流触发。接着，第 4.2.2节对训练过程中参数聚合的通信通道的实现进行说明，包括工作节点的模型参数分片与传输，以及外部存储服务的选择。最后，第 4.2.3节对如何解决函数最大生命周期限制对长时间训练的影响进行了说明。

4.2.1　训练初始化

在无服务器计算范式下，无服务器计算函数作为工作节点开展分布式训练。对于开发者提交的训练负载，在执行分布式训练之前，需要先进行一系列的初始化工作，包括确定 K-REDUCE 通信模式下的参数配置，并依据参数划分训练数据集，然后触发对应的函数开展分布式训练。在训练前的初始化阶段，本文通过运行一个函数作为触发节点 *Trigger* 执行上述操作。

4.2.1.1　训练参数配置

开发者提交的训练负载包括训练模型 *model*、训练数据集 *dataset*、训练总轮次 E、端到端的训练时间限制 T_{max} 以及最大全局批次大小限制 B^g_{max}。触发节点 *Trigger* 首先借助负载分析模块 *Profiler* 对训练模型进行分析，确定对应的训练系数。然后，触发节点 *Trigger* 基于两阶段启发式搜索算法 3.2确定 K-REDUCE 通信模式下的五个参数，包括函数的内存配额 M、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n。

4.2.1.2　训练数据集划分

在确定了工作节点的数量 W 和聚合节点的数量 K 后，工作节点被划分为聚合节点和非聚合节点两类，同时确定了两者的训练批次大小 B^a 和 B^n。接下来，触发节点 *Trigger* 基于两类工作节点的数量及对应的批次大小，按比例划分训练数据集。具体地，对于 D 个训练样本，聚合节点分配的训练样本数 D^a 如式 4-3所示，非聚合节点分配的训练样本数 D^n 如式 4-4所示。

$$D^a = D \cdot \frac{B^a}{B^g} = D \cdot \frac{B^a}{B^a \cdot K + B^n \cdot (W - K)} \tag{4-3}$$

$$D^n = D \cdot \frac{B^n}{B^g} = D \cdot \frac{B^n}{B^n \cdot K + B^n \cdot (W - K)} \tag{4-4}$$

触发节点 *Trigger* 在本地完成训练数据集的划分后，将各部分进行编号，并上传至 AWS S3 指定的桶（bucket）中。同时，触发节点 *Trigger* 也上传训练模型至 AWS S3 中用于后续的训练。由于训练数据集和训练模型的规模较大，因此对其进行压缩后再上传，并由对应的工作节点下载到本地后进行解压。

4.2.1.3　训练工作流触发

在完成了上述初始化工作后，触发节点 *Trigger* 将并行触发多个工作节点开展训练。根据参数配置阶段确定对应函数的内存配额数量 M，然后触发对应数量 W 的工

作节点。同时，触发节点 *Trigger* 通过环境变量向各工作节点 *Worker* 传递配置参数，包括工作节点的编号 i、工作节点的数量 W、聚合节点的数量 K、聚合节点的批次大小 B^a 和非聚合节点的批次大小 B^n。工作节点的编号 i 从 0 至 $W-1$。其中，编号小于 K 的工作节点作为聚合节点 *Aggregator*，以批次大小 B^a 开展训练，聚合阶段采用同步更新策略。而编号大于等于 K 的工作节点作为非聚合节点 *Non-aggregator*，以批次大小 B^n 开展训练，聚合阶段采用异步更新策略。触发节点 *Trigger* 将划分后的训练数据集对应编号并上传，各工作节点被触发后从 AWS S3 中下载对应编号的训练数据集。

4.2.2　通信通道实现

由于无服务器计算函数的无状态特性，函数间无法直接进行网络通信。为此，通信阶段以聚合节点作为聚合操作的执行单位，并以外部存储服务作为函数间通信的存储媒介，从而实现了一个高效的通信通道。本节中分别从工作节点侧和外部存储侧说明通信通道的具体实现。

4.2.2.1　模型参数分片与传输

在一轮迭代中，各工作节点执行完对应批次的本地训练后，进入到通信阶段进行参数聚合。在上传阶段之初和下载阶段之后，各工作节点需要分别进行模型参数的分片和重组。由于深度训练模型的结构较为复杂，因此在进行模型参数的分片之前以及重组后需要进行序列化和反序列化操作。在序列化操作中，工作节点先将模型参数按照层级转化为一个一维的向量，然后根据环境变量中聚合节点的数量将参数向量平均划分为 K 个分片。而在反序列化操作中，工作节点将聚合后的 K 个参数分片重新拼接为一维的向量，然后按照训练模型的结构重组为模型参数。

在通信阶段的传输中，各工作节点的参数分片按照统一规则编号，从而确保参数聚合的一致性。在上传阶段中，对于第 l 轮迭代，编号为 i 的工作节点将第 j 个参数分片编号为 $S_{l\_i\_j}$。在聚合阶段中，编号为 t 的聚合节点拉取对应迭代 l 下所有满足 $j=t$ 的参数分片到本地进行聚合，然后再将聚合后的参数分片编号为 A_{l_t} 上传至外部存储服务中。最后，在下载阶段中，聚合节点和非聚合节点分别采用不同的同步策略获取聚合后的参数分片。对于聚合节点，其下载第 l 轮迭代下的所有 K 个聚合参数分片 A_{l_t}。而对于非聚合节点，其下载第 $l-1$ 迭代下的所有 K 个聚合参数分片 A_{l-1_t}。

4.2.2.2　外部存储服务选择

外部存储服务作为通信阶段中函数间的通信媒介，需要提供以下抽象能力。首先，外部存储服务需要存储工作节点在各迭代中的参数分片，并提供按名称的高效检索。此外，外部存储服务需要为工作节点提供简单易用的上传和下载对象的 API 接口。为了保证分布式训练过程中参数聚合的高效，外部存储服务还需要提供较高的并发性和吞吐量。对于满足上述抽象能力的外部存储服务，FasDL 可以根据训练任务的需求快速兼容。本文选择了对象存储服务如 AWS S3 和缓存服务 AWS ElasticCache，并对两者作为通信通道的实现进行了说明和对比。

对象存储服务如 AWS S3 将数据存储为对象，提供了数据的持久性存储能力。同时，AWS S3 提供了高度可扩展性，能够轻松应对大量数据的存储和高并发请求。在数据访问上，AWS S3 提供了简易的对象访问接口，且数据访问的成本较低，但同时其数据访问的速度也相对较慢。AWS S3 提供了上传对象的接口 `get_object` 和下载对象的接口 `put_object`，能够按照指定的存储桶名称以及对象键对对象进行操作。另外，AWS S3 提供了按照名称检索对象的接口 `list_objects_v2`。

缓存服务如 AWS ElasticCache 提供了高性能、低延迟的内存数据库。数据存储在内存中，能够加快数据访问速度，显著提高数据访问的性能和响应能力。相对地，AWS ElasticCache 的数据访问的成本较高。在缓存服务中，数据以键值对（$key-value$）的形式存储，提供了上传数据的接口 `set` 和下载数据的接口 `get`。同时，能够直接通过键进行高效的数据检索和访问。

第 5.5节中对比了不同外部存储服务的性能和开销。本文选择 AWS S3 和 AWS ElasticCache 作为对象存储服务和缓存服务的代表，在不同训练负载下测试了训练的端到端时间和训练开销，并对两者作为通信通道的优劣进行了对比和分析。

4.2.3　函数最大生命周期限制处理

在分布式训练过程中，训练任务流需要在多个训练轮次下持续进行。然而，无服务器计算平台对计算函数的最大生命周期进行了限制。例如，在 AWS Lambda 中，Lambda 函数的最大生命周期为 15 分钟。因此，需要额外引入策略解决函数最大生命周期对持久训练的限制。

本文借鉴了 LambdaML 中的分层调用策略（hierarchical invocation mechanism）解决函数最大生命周期的限制。在触发节点 *Trigger* 触发了训练工作流后，各工作节点负责完成第一轮次的训练任务。在该轮次的训练完成后，各工作节点将本轮训练后

的训练模型上传至 AWS S3 中作为检查点（checkpoint），然后各自触发自身（trigger self）以启动下一轮次的训练。具体地，每个工作节点将模型按照节点编号进行标注并上传，然后触发新的计算函数下载对应的训练模型，以继续下一轮次的训练。新触发的计算函数继承了原计算函数的所有环境变量以及模型参数，因此，本文提出的训练架构可以在多轮训练中保持训练工作流的连续性以及训练结果的一致性。

4.3 本章小结

本章对无服务器计算范式下分布式训练架构的实现细节进行说明。首先，本章介绍了 FasDL 架构中负载分析模块的实现方案，包括对训练模型特征和无服务器计算平台性能的分析。其次，本章介绍了基于 K-REDUCE 通信模式的训练框架的实现细节，包括初始化阶段中触发节点的实现、基于计算函数和外部存储的通信通道的实现以及应对函数最大生命周期限制的处理策略。基于上述分布式训练架构实现方案，本文实现了一个 FasDL 架构的原型，用于下一章的性能验证实验。

第 5 章 实验与分析

本章基于构建的 FasDL 原型在 AWS Lambda[37] 平台上开展实验，以验证训练架构的性能。首先，第 5.1节介绍了实验的配置，包括训练负载以及负载分析结果。然后，第 5.2节对第 3.3节中训练性能的建模精度进行验证，包括端到端的训练时间以及训练开销两个预测模型。接着，第 5.3节对第 3.4节中的两阶段启发式搜索算法的性能进行验证，包括搜索算法的搜索效率以及搜索结果与最优解之间的偏差。接下来，第 5.4节将第 3.2节中提出的 K-REDUCE 通信模式与过往工作进行对比，分别从端到端训练时间、训练开销以及收敛效率三个维度进行分析。此外，第 5.5节对比了基于不同存储服务的通信通道实现下 K-REDUCE 的性能表现，并将其与基于服务器的通信方案进行对比。最后，第 5.6节对比了 FasDL 下分布式训练与单机训练的性能表现。

5.1 实验配置

表 5.1 训练负载
Table 5.1 Workloads

训练模型	数据集	模型大小(MB)	训练性能相关系数		
			a	b	m
BERT-Base	CoLa	417.17	2344.97	2.37	1145.66
ResNet50	CIFAR10	97.49	37.19	12.48	-111.46
MobileNet	CIFAR10	13.37	13.50	9.57	-92.88
SqueezeNet	CIFAR10	4.71	2.16	22.60	-93.22

首先，对 FasDL 训练框架所使用的无服务器计算平台、外部存储服务以及训练负载进行说明。本文在 AWS Lambda 上构建了 K-REDUCE 通信模式下的训练工作流。同时，本文选择 AWS S3[33] 和 AWS ElasticCache[34] 作为通信通道中的外部存储服务。训练负载选择了 SqueezeNet[52]、MobileNet[61]、ResNet50[51] 和 BERT-Base[62] 四个模型。前面三个训练模型使用 CIFAR-10[63] 作为训练数据集，而 BERT-Base 训练模型使用 CoLA[64] 作为训练数据集。同时，本文使用第 4.1节中的负载分析器 *Profiler* 对上述四个训练模型和 AWS Lambda 训练平台的特性进行分析。表 5.1列出了分析得出

的四个训练模型的模型大小以及与训练性能相关的系数 a、b 和 m。另外，图 5.1和图 5.2分别展示了 AWS Lambda 与 S3 之间上行和下行吞吐量的拟合结果。

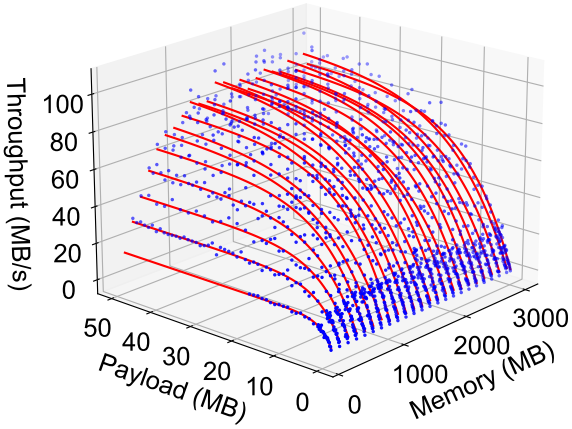


图 5.1　AWS Lambda 和 S3 之间上行吞吐量拟合结果
Figure 5.1　Fitting of Upload Throughput between AWS Lambda and Amazon S3

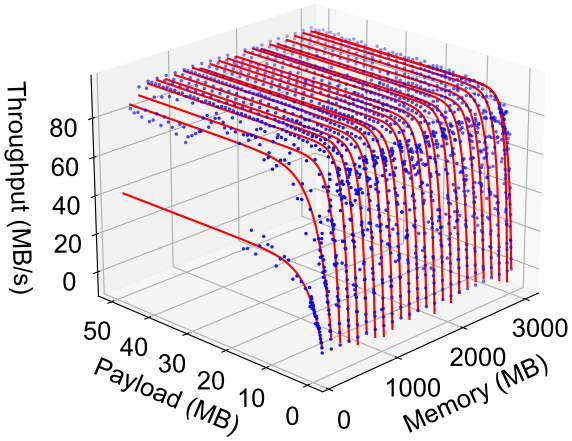


图 5.2　AWS Lambda 和 S3 之间下行吞吐量拟合结果
Figure 5.2　Fitting of Download Throughput between AWS Lambda and Amazon S3

接下来，对实验对比中使用的基准（benchmark）进行说明。为了验证对训练性能的建模的准确性，本文将对端到端训练时间和训练开销的预测值与实际值进行对比。为了验证两阶段启发式算法的性能，本文将其与暴力搜索算法进行对比，包括参数搜索用时和搜索结果精度两个方面。在验证 K-REDUCE 通信模式的性能时，本文选择 LambdaML[14] 中的 AllReduce 和 ScatterReduce 通信模式作为对比基准。

由于 LambdaML 并非在无服务器计算范式下提出，因此本文构建了 AllReduce 和 ScatterReduce 在无服务器计算范式下的实现方案。在对比不同通信通道实现方案的性能时，本文对比了 AWS S3 和 AWS ElasticCache 两种存储服务构建的通信通道的性能。同时，本文将 λDNN[13] 作为对比基准，对比基于存储的通信通道的性能与基于参数服务器的通信通道的性能。最后，本文实现了各训练负载的单机训练，并与 FasDL 训练框架下的分布式训练进行对比。

5.2 建模预测性能验证

本节对第 3.3节中构建的系统性能建模的预测性能进行分析。负载分析器 *Profiler* 对四个模型负载的训练性能的分析结果如表 5.1中的训练系数 a、b 和 m 所示。接下来，本文采用两阶段启发式搜索算法搜索出各个训练负载对应的参数配置，并基于第 3.3节中构建的系统性能建模计算出各个负载的端到端训练时间和训练开销的预测值，然后将其与实际运行中测量的实际值进行对比，结果如图 5.3所示。

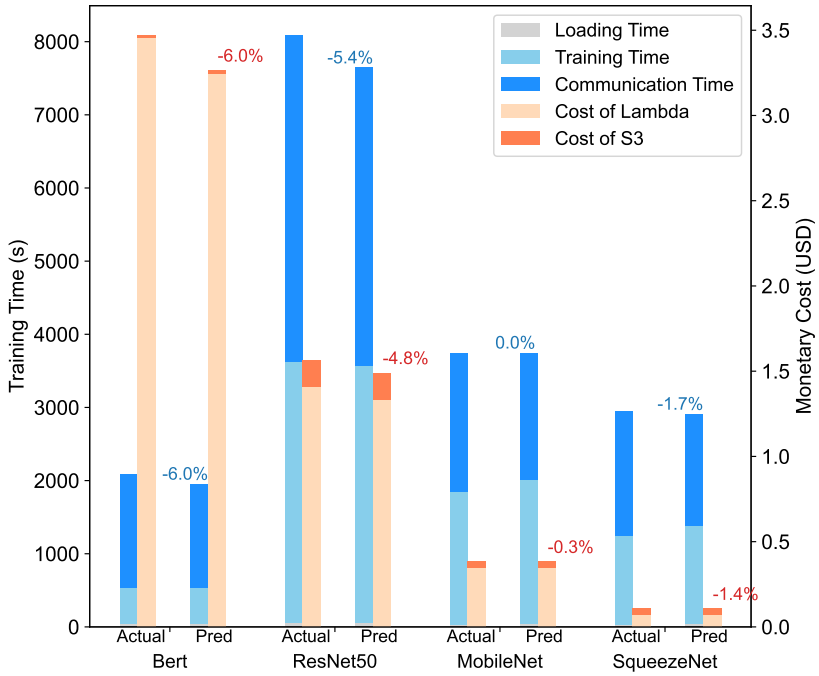


图 5.3 K-REDUCE 通信模型下训练性能的预测精度
Figure 5.3 Precision Analysis of Predictions for K-REDUCE

图 5.3中标注了各个训练负载下，训练时间和训练开销的预测值与实际运行值之

间的误差。性能建模对训练时间和训练开销的预测误差最大不超过 6%，对于 MobileNet 和 SqueezeNet 模型，训练时间和训练开销的预测误差不超过 2%。端到端的训练时间分解为加载时间（Loading Time）、训练时间（Training Time）和通信时间（Communication Time）。而对于训练开销，分解为 Lambda 函数的开销（Cost of Lambda）和 S3 服务的开销（Cost of S3）。从图中可见，训练开销的误差主要来自于 Lambda 函数的开销，而 Lambda 函数的计费误差主要来源于对函数运行总用时的预测误差。而端到端训练时间的预测误差则主要来自于对通信时间的预测误差，该误差源于实际运行时函数的通信吞吐量的波动。由于通信相关的系数是在规模受限且带有噪音的样本中通过最小二乘法拟合得出的，因此最终导致了对通信时间预测的误差。总体而言，性能建模模块对训练性能的建模能够准确地预测实际训练的表现，其预测精度可以达到 94%，从而为后续系统参数配置的搜索提供了基础和保障。

5.3　参数配置性能验证

本节对基于剪枝策略的两阶段启发式算法的参数配置优化模块的性能进行验证。该优化器借助参数范围剪枝来缩小参数的搜索范围，然后采用两阶段搜索减少参数组合搜索空间，以加快搜索进程。参数范围剪枝将 NP 难问题的开销降低到可接受的水平，并消除了因诸如执行内存不足等限制而导致的不可行的参数配置。对于参数配置优化模块的参数搜索效果，本文从参数搜索效率和参数搜索结果的精度两方面进行评估。本文选择暴力搜索（Brute-force）算法作为基准，即在参数范围剪枝后对五个参数进行暴力搜索。针对表 5.1中的四个训练负载，本文分别使用暴力搜索算法和两阶段启发式算法进行参数配置，两类参数搜索策略的性能对比结果如表 5.2所示。

表 5.2　不同搜索策略的性能对比
Table 5.2　Comparison of Different Search Strategies

训练模型	参数搜索策略	预测的训练开销(USD)	性能衰退程度	搜索用时开销(s)	搜索效率提升倍数
BERT-Base	两阶段启发式搜索	3.42	11.6%	32.1	4.8x
	暴力搜索	3.02		155.0	
ResNet50	两阶段启发式搜索	1.49	14.6%	290.5	9.1x
	暴力搜索	1.30		2645.6	
MobileNet	两阶段启发式搜索	0.38	2.7%	83.1	11.4x
	暴力搜索	0.37		954.6	
SqueezeNet	两阶段启发式搜索	0.104	6.1%	217.0	2.0x
	暴力搜索	0.098		443.9	

相比于暴力搜索算法，两阶段启发式搜索算法之多将搜索速度提高了 11.4 倍。

这种高效率是通过将参数配置划分为两个独立阶段来实现的，极大地降低了搜索空间的范围和搜索算法的复杂度。另外，为了验证两阶段搜索算法搜索结果的有效性，本文对比其搜索得到的近优解与暴力搜索算法得到的最优解在训练开销上的差异。表 5.2中搜索结果的性能衰退程度 $decay$ 的定义如式 5-1 所示。

$$decay = (\frac{Cost_{\text{Two-Stage}}}{Cost_{\text{Brute-force}}} - 1) \times 100\% \tag{5-1}$$

如表 5.2所示，两阶段启发式搜索算法在四个训练负载上搜索结果性能的衰退程度分别为 11.6%、14.6%、2.7% 和 6.1%。借助两阶段启发式搜索算法，参数配置优化模块用搜索结果性能的轻微下降换取搜索效率的显著提升。

5.4 K-REDUCE通信模式优化性能验证

为了验证 K-REDUCE 通信模式的性能表现，本文将其与 LambdaML 中的 AllReduce 和 ScatterReduce 通信模式进行综合的对比和分析，包括端到端的训练时间、训练开销以及收敛性能三个方面。本节实验采用 AWS S3 作为通信通道中的外部存储服务，并采用两阶段启发式搜索算法为四个训练负载进行参数配置，对应的约束条件和参数配置结果如表 5.3所示。

表 5.3 不同通信模式下的参数配置结果

Table 5.3 Parameter Configuration of Different Communication Patterns

训练模型	训练时间约束(s)	全局批次大小约束	K-REDUCE				ScatterReduce				AllReduce			λDNN		
			N	M	B^a	K	B^n	N	M	B	N	M	B	N	M	B
BERT-Base	2500	640	10	10240	32	4	84	15	10240	32	No Fit			10	10240	32
ResNet50	8000	1024	7	1536	128	4	170	8	1664	128				2	3328	512
MobileNet	4000	512	3	1920	128	1	192	4	2048	128	4	1920	128	2	2304	256
SqueezeNet	3000	384	2	768	128	1	201	2	1792	128	3	640	128	2	768	192

5.4.1 训练时间和训练开销性能对比

为了验证 K-REDUCE 通信模式在优化训练时间和训练开销上的表现，我本节从两个方面进行对比和分析。一方面，对不同通信模式在优化问题式 3-31下的表现进行对比，包括 K-REDUCE（*K-Opt*）、AllReduce（*All-Opt*）和 ScatterReduce（*Scatter-Opt*）三个通信模式。三者在优化问题下的参数配置如表 5.3所示。另一方面，在相同的资源配置下对比三种通信模式的训练性能。具体地，采用 K-REDUCE 通信模式对应的工作节点数量 W、函数内存配额 M 和训练批次大小 B^a 在 AllReduce（*All-CP*）和 ScatterReduce（*Scatter-CP*）通信模式下开展训练，作为对照组（counterpart）对

比其训练时间和训练开销。四个训练负载下的性能对比结果如图 5.4、图 5.5、图 5.6 和图 5.7 所示。

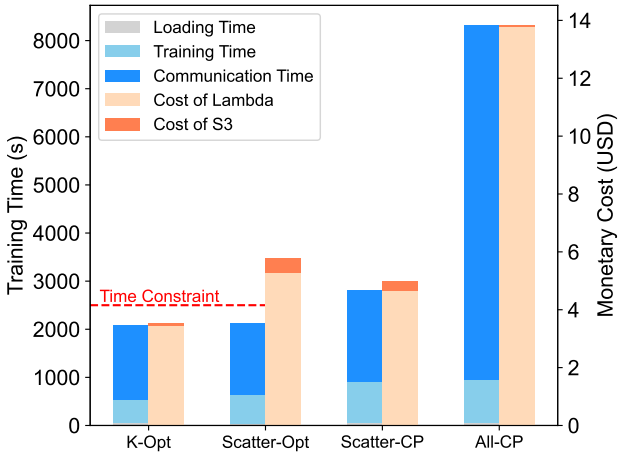


图 5.4　BERT 模型下训练时间和训练开销的对比

Figure 5.4　Comparison on End-to-End Training Time and Monetary Cost of BERT

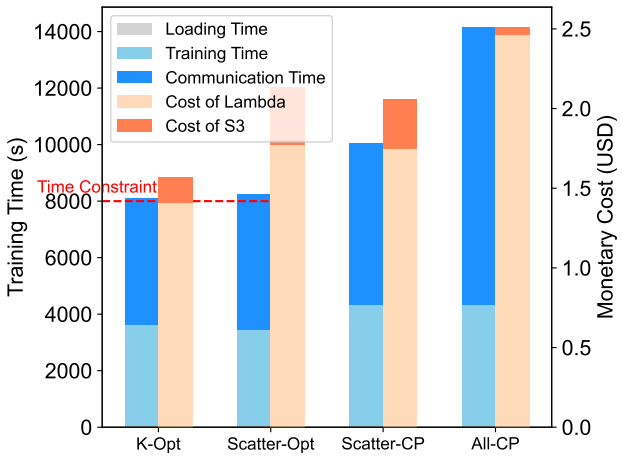


图 5.5　ResNet50 模型下训练时间和训练开销的对比

Figure 5.5　Comparison on End-to-End Training Time and Monetary Cost of ResNet50

5.4.1.1　不同通信模式在优化问题下的性能对比

第 3.4.1节提出了目标优化问题式 3-31，以在给定的最大训练时间约束和最大全局批次大小约束下，最小化训练的开销计费。本节对比该优化问题下 K-REDUCE 通信模式与 ScatterReduce 和 AllReduce 通信模式的性能表现。四个训练负载在优化问题中的性能约束以及参数配置如表 5.3所示。对于 K-REDUCE 通信模式，其最优参

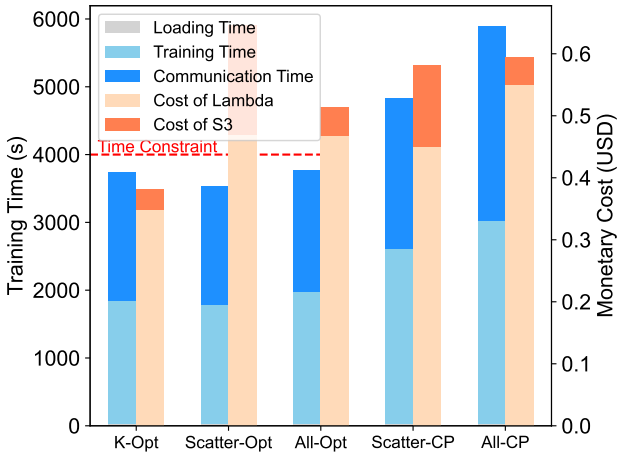


图 5.6 MobileNet 模型下训练时间和训练开销的对比

Figure 5.6 Comparison on End-to-End Training Time and Monetary Cost of MobileNet

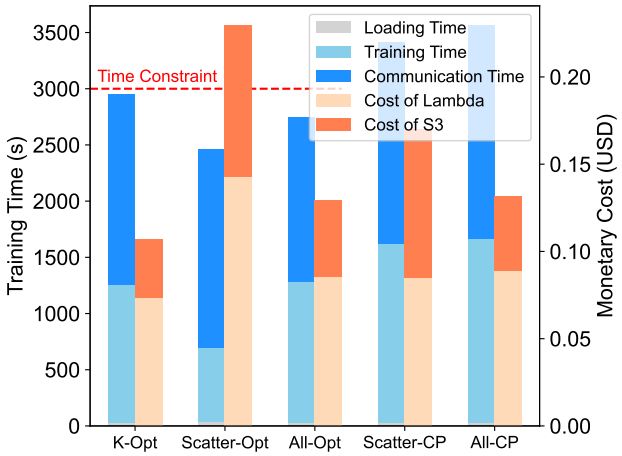


图 5.7 SqueezeNet 模型下训练时间和训练开销的对比

Figure 5.7 Comparison on End-to-End Training Time and Monetary Cost of SqueezeNet

数配置 *K-Opt* 通过两阶段启发式搜索算法确定。对于 ScatterReduce 和 AllReduce 通信模式，其最优参数配置 *Scatter-Opt* 和 *All-Opt* 通过暴力搜索算法确定。因此，实际是在比较 FasDL 框架下 K-REDUCE 通信模式的性能与 AllReduce 和 ScatterReduce 通信模式的最优性能。其中，对于 BERT-Base 模型和 ResNet50 模型，没有任何参数配置能够在给定的训练时间约束下，通过 AllReduce 通信模式完成训练。这是由于对于上述规模较大的训练模型，在通信阶段仅选择一个工作节点作为聚合节点导致了较高的通信时间开销。此时，唯一的聚合节点成为了分布式训练的性能瓶颈。

如图 5.4、图 5.5、图 5.6 和图 5.7 所示，在满足训练时间约束和全局批次大小

约束的条件下，K-REDUCE 的最优配置 *K-Opt* 在训练开销上均优于 ScatterReduce 的最优配置 *Scatter-Opt* 和 AllReduce 的最优配置 *All-Opt*。与 ScatterReduce 通信模式相比，K-REDUCE 在 BERT-Base、ResNet50、MobileNet 和 SqueezeNet 四个模型上分别降低了 39.0%、26.7%、25.8% 和 53.2% 的训练开销。与 AllReduce 通信模式相比，K-REDUCE 在 MobileNet 和 SqueezeNet 两个模型上分别降低了 25.8% 和 17.6% 的训练开销。

K-REDUCE 通信模式通过选择最优数量的聚合节点以降低通信开销，同时通过不均等的数据划分和混合异步并行协议（HAP）进一步加速训练过程，从而在整体上提升了训练的效率。如表 5.3所示，在相同的训练时间约束下，K-REDUCE 通信模式的资源配置相比 ScatterReduce 和 AllReduce 通信模式使用了更少的资源（如工作节点的数量 W 和函数的内存配额 M）。由此，K-REDUCE 通信模式显著减少了部署和训练的开销计费。

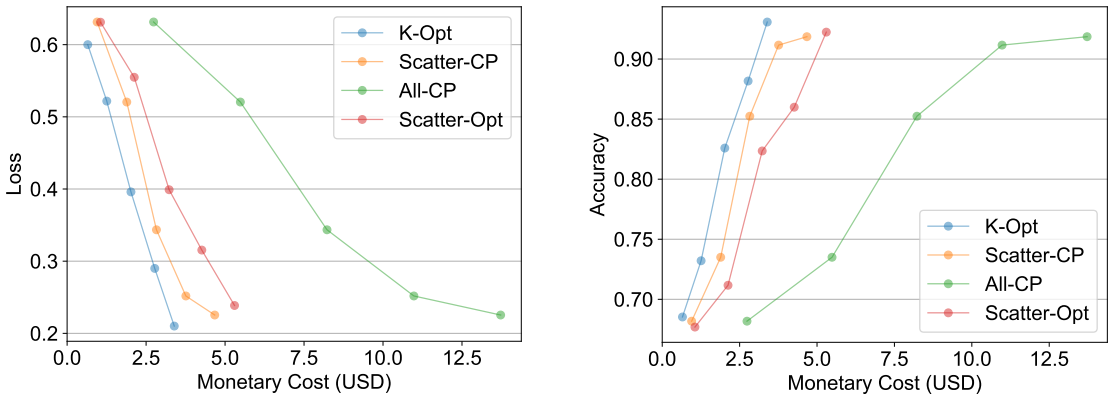
5.4.1.2　不同通信模式在固定资源配置下的性能对比

本节进一步对比 K-REDUCE 通信模式与 ScatterReduce 和 AllReduce 通信模式在相同的资源配置下的性能表现，以观察 K-REDUCE 通信模式在训练时间和训练开销上的增益效果。本节实验设置了与 *K-Opt* 工作节点数量 W、内存配额 M 和批次大小 B 相同的两个对照组（counterpart）*All-CP* 和 *Scatter-CP*，分别采用 AllReduce 和 ScatterReduce 通信模式开展训练。如图 5.4、图 5.5、图 5.6 和图 5.7 所示，对于四个训练负载，*K-Opt* 在端到端的训练时间和训练开销上均优于 *All-CP* 和 *Scatter-CP*。平均而言，K-REDUCE 相比 ScatterReduce 在训练时间上提升了约 16.8%，在训练开销上提升了约 28.3%。而相比 AllReduce，K-REDUCE 通信模式在训练时间上提升了约 33.6%，在训练开销上提升了约 31.4%。

在相同的资源配置下，K-REDUCE 通信模式通过两方面的优化降低了端到端的训练时间。一方面，K-REDUCE 通信模式在给定的工作节点数量下选择了最优的聚合节点数量 K，从而最小化了通信时间。另一方面，K-REDUCE 利用了非聚合节点在聚合阶段中的空闲 CPU 资源用于额外训练数据集的训练，从而进一步加速模型的训练速度。由于 Lambda 函数的计费与函数的运行时间成正比，因此随着端到端训练时间的减少，Lambda 函数的计费也随之减少。此外，相比于 ScatterReduce 通信模式利用所有 W 个工作节点参与模型聚合，K-REDUCE 通信模式选择了更少数量的聚合节点，从而减少了聚合阶段的通信请求数量。由于 AWS S3 对网络请求的计费基于请求的数量，因此 K-REDUCE 通信模式进一步减少了通信阶段的开销计费。综上分

析，K-REDUCE 通信模式借助上述优化策略同时取得了训练时间和训练开销两方面的增益。

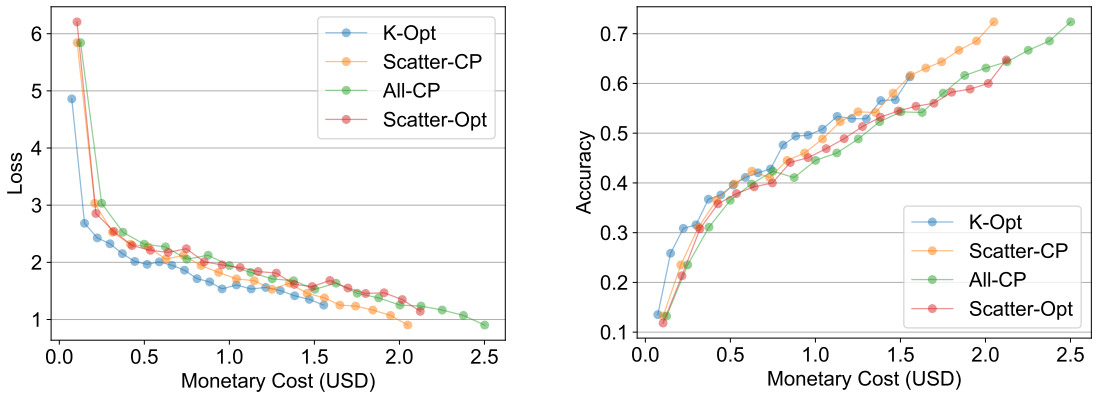
5.4.2 收敛性能对比



(a) Loss of BERT (b) Accuracy of BERT

图 5.8 BERT 模型收敛效率的对比

Figure 5.8 Comparison on Convergence Efficiency of BERT

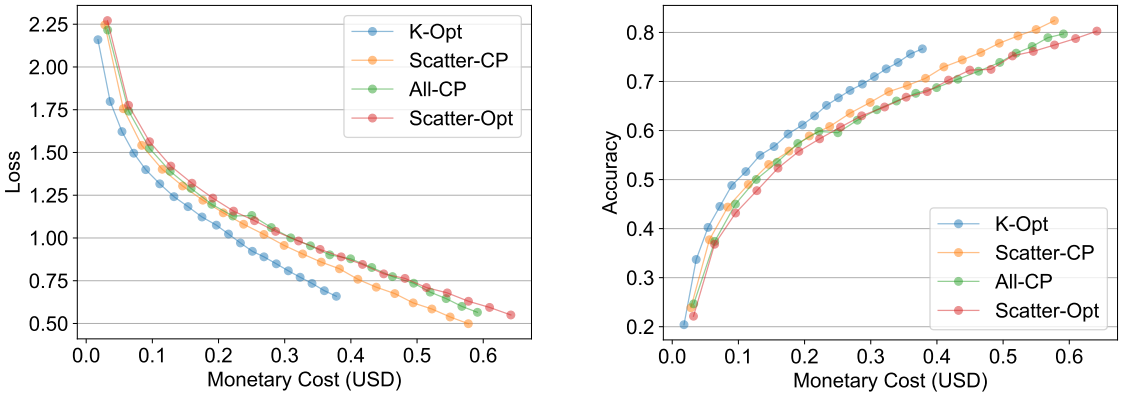


(a) Loss of ResNet50 (b) Accuracy of ResNet50

图 5.9 ResNet50 模型收敛效率的对比

Figure 5.9 Comparison on Convergence Efficiency of ResNet50

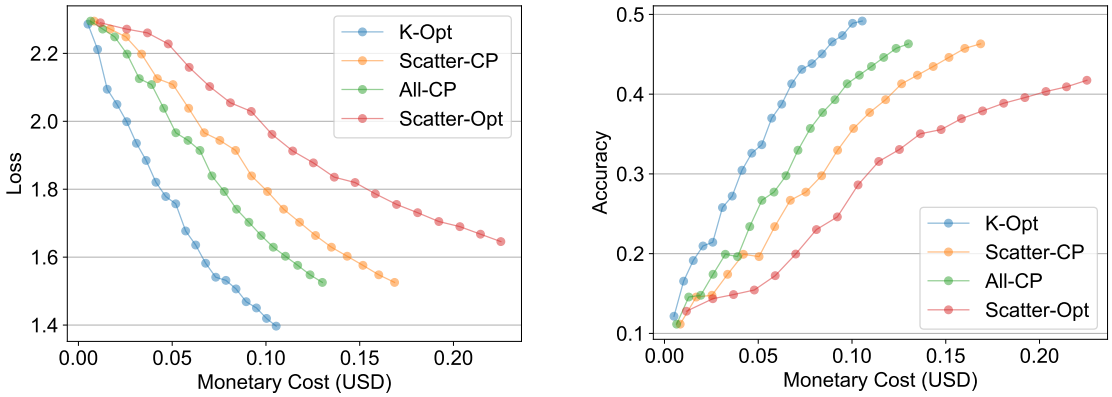
相比 AllReduce 和 ScatterReduce 通信模式采用批同步并行（Bulk Synchronous Parallel）协议，K-REUDCE 通信模式采用了混合异步并行（Hybrid Asynchronous Parallel）协议实现连续迭代中的参数聚合。在相同的训练开销下，四个训练负载的训练



(a) Loss of MobileNet　　(b) Accuracy of MobileNet

图 5.10　MobileNet 模型收敛效率的对比

Figure 5.10　Comparison on Convergence Efficiency of MobileNet



(a) Loss of SqueezeNet　　(b) Accuracy of SqueezeNet

图 5.11　SqueezeNet 模型收敛效率的对比

Figure 5.11　Comparison on Convergence Efficiency of SqueezeNet

损失（training loss）和准确率（accuracy）的变化情况如图 5.8、图 5.9、图 5.10和图 5.11所示。

混合异步并行协议控制聚合节点和非聚合节点之间的过时度（staleness）为 1，从而确保了 K-REDUCE 通信模式下的收敛效率。图中展示了在最优配置下 *Scatter-Opt* 和 *All-Opt* 的收敛表现以及在相同资源配置下 *Scatter-CP* 和 *All-CP* 的收敛表现。在大部分情况下 K-REDUCE 通信模式的收敛性能的表现均优于 ScatterReduce 和 AllReduce 通信模式的表现。

5.5 通信通道实现方案性能比较

为了解决无服务器计算函数在分布式训练中无法直接通信的问题，共有两种常见的构建通信通道的实现方式，分别是基于存储服务的通信通道和基于参数服务器的通信通道。FasDL 架构采用了基于存储服务的通信通道，而 λDNN 则采用了一个独立的参数服务器执行参数聚合。通过负载分析模块和性能建模模块，FasDL 可以轻易地兼容不同类型的存储服务构建通信通道，包括持久的对象存储服务（如 AWS S3）和缓存服务（如 AWS ElasticCache）。

本节首先对比 FasDL 训练架构下采用不同类型的存储服务构建通信通道时的性能表现。本节在 FasDL 原型中分别使用 AWS S3 和 AWS ElasticCache 实现了通信通道并测试了 FasDL 训练架构的性能表现。此外，本节还参照 λDNN 实现了基于参数服务器的分布式训练框架，并将其与 FasDL 的训练性能进行对比，从而观察两类通信通道实现方式的性能差异。综上，本节在四个训练负载上分别实现了上述三种通信通道，并测量了其端到端的训练时间以及训练开销，结果如图 5.12所示。

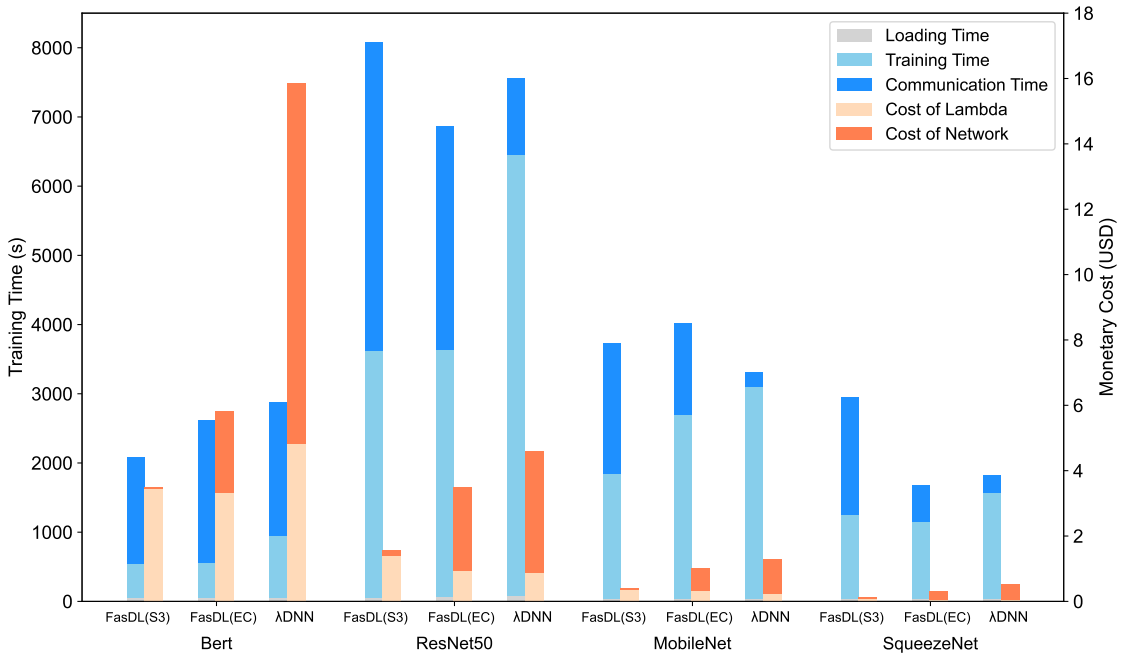


图 5.12 不同通信通道实现在时间约束下最小化训练开销的性能对比

Figure 5.12 Comparison of Different Communication Channel Implementations for Minimized Cost under Time Constraints

5.5.1　持久对象存储服务和缓存服务的性能对比

在给定的端到端最大训练时间和最大全局批次大小的约束下，本节将 FasDL 训练框架结合 AWS S3 和 AWS ElasticCache 对表 5.1中的四个训练负载进行最优的参数配置，以最小化训练的开销。各个负载的实际训练时间以及最小化的训练开销如图 5.12所示。

由于 AWS ElasticCache 的开销计费与传输的分片大小相关，因此其通信开销随着训练模型规模的增大而上涨。因此对于复杂的训练模型，借助 AWS ElasticCache 进行函数间通信时，其通信开销的计费将远大于使用 AWS S3 服务。同时，由于 AWS ElasticCache 的数据读取和存储速度高于 AWS S3，因此其总体的通信时间在大部分情况下更少。然而，当训练一些较为复杂的模型（如 BERT 模型）时，使用 AWS ElasticCache 作为通信通道时其训练时间和训练开销均劣于 AWS S3 的表现。如式 3-28所示，当训练模型的规模较为庞大时，AWS ElasticCache 的计费开销 C_{EC} 随着工作节点数量 W 的增加快速上涨。因此，对于该类模型，其最优的参数配置倾向于配置更少的工作节点参与训练。此外，函数内存配额 M 的最大可配置值限制了工作节点的批次大小。因此对于复杂的训练模型，其全局批次大小 B^g 无法充分利用最大全局批次大小的限制，进而导致需要更多轮的迭代和通信。

5.5.2　基于存储服务与基于参数服务器的通信通道的性能对比

本节对比 FasDL 训练框架与 λDNN 训练框架的性能，其中 λDNN 使用一个独立的参数服务器作为参数聚合的通信通道。本节按照 λDNN 的实现方式，使用一个 EC2 实例（m5.large，2 vCPUs 和 8 GB 内存）作为参数服务器（parameter server），并在相同的优化问题式 3-31下开展实验。

两者的性能对比结果如图 5.12所示。与 AWS ElasticCache 服务相似，EC2 实例的网络通信计费也与传输的分片大小相关。因此，使用 EC2 实例开展训练时，分布式训练的训练时间和训练开销与使用 AWS ElasticCache 时的表现相似。且由于 EC2 实例的计费成本更高，因此在四个任务负载上 λDNN 训练框架的总开销计费均高于 FasDL 训练框架。总体而言，不论是使用 AWS S3 实现通信通道或是使用 AWS ElasticCache 实现通信通道，FasDL 训练框架相比 λDNN 均有更优的性能表现。在最优情况下，FasDL 训练框架相比 λDNN 节省了 78.13% 的训练开销。此外，对比基于参数服务器的通信通道实现，对象存储服务和缓存服务采用了按使用计费（pay-as-you-go）的模式，且同样无需进行额外配置（serverless）。因此，使用基于存储服务

的通信通道实现能够进一步地减少训练开销并减轻开发者的部署压力。

5.6 FasDL与单机训练比较

本节对比了 FasDL 训练框架与单机训练在最小化端到端训练时间上的表现。本节在 AWS Lambda 平台上通过触发一个单独的 Lambda 函数执行本地训练。由于单机训练不涉及多个工作节点间的参数聚合，因此训练过程中运行函数仅执行模型训练。当前无服务器计算平台对单个函数的最大资源配置进行了限制，例如对于 AWS Lambda 平台，其 Lambda 函数的最大内存配额为 10240MB。由于单机训练无需额外的通信机制进行参数聚合，其训练开销要优于分布式训练框架。因此，本节为执行本地训练的 Lambda 函数分配最大的内存配额，以观察单机训练在最小化训练时间上的表现。而对于 AWS S3 和 AWS ElasticCache 存储服务下的训练，本节相应地搜索最小化端到端训练时间的参数配置并执行训练。两者的训练结果如图 5.13所示，图中列出了四个训练负载下的最小化的端到端训练时间以及对应的训练开销。

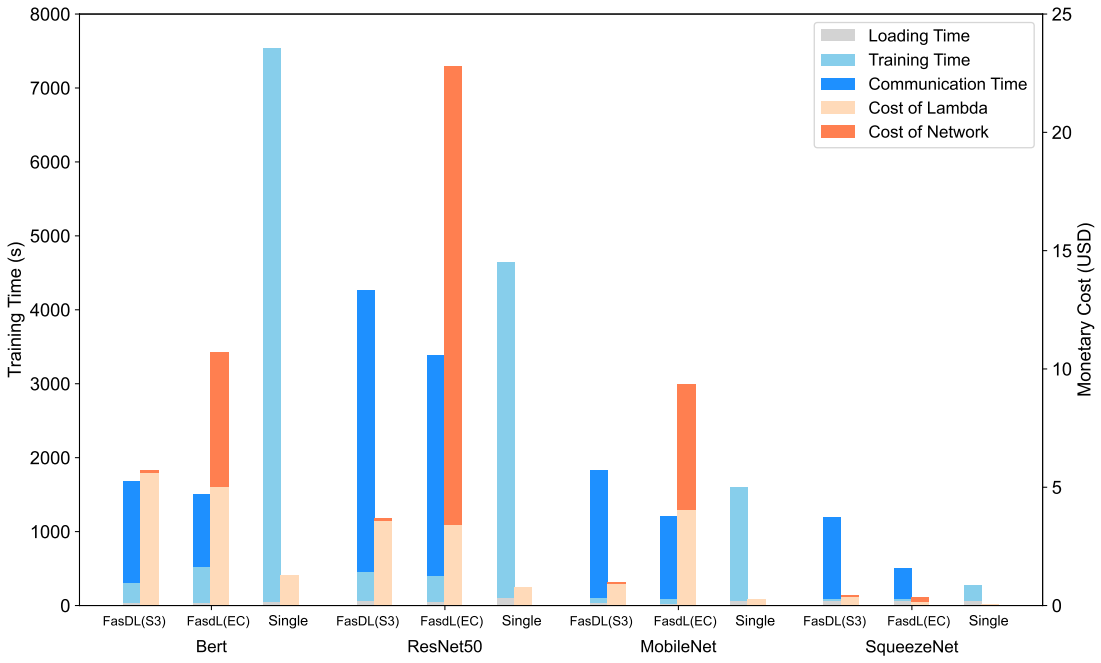


图 5.13 最小化端到端训练时间下 FasDL 框架训练与单机训练的性能对比

Figure 5.13 Comparison Between FasDL and Standalone Implementations for Minimized End-to-End Training Time

如图所示，由于单机训练不涉及额外的通信开销，因此在训练模型较小（如 SqueezeNet 模型）时，其在训练时间和训练开销的表现上均优于 FasDL 训练框架。

然而，随着训练模型变得更为复杂，无服务器计算平台对单个函数的最大内存配额的限制成为了训练的性能瓶颈。例如，对于 BERT 训练模型，其在单机训练下的训练时间达到了 FasDL 训练框架的五倍。总体而言，无服务器计算范式下的单机训练由于按使用量计费（pay-as-you-go）的计费模式，能够显著地减少训练开销。然而，单个函数的资源限制使得在训练复杂模型时，单机训练方式无法进一步地提升训练速度。

5.7　本章小结

本章在 FasDL 训练架构上开展对比实验，对 FasDL 架构中的三个核心模块的性能分别进行验证。实验结果表面，在训练性能的预测上，性能建模模块的预测误差控制在 6% 以内。在参数自动化配置上，参数配置优化模块能够以微小的性能损失换取较大的搜索效率提升。对比 LambaML 的通信模式实现，K-REDUCE 通信模式在训练速度上提升了 16.8%，训练成本降低了 28.3%，同时在大部分情况下取得了更优的收敛性能。另外，FasDL 训练架构可以兼容不同存储服务的通信通道的实现，相比基于参数服务器的 λDNN，在最优情况下节省了 78.13% 的训练开销。

第 6 章 结论与展望

6.1 全文总结

本文提出了一个无服务器计算范式下的高性能分布式训练框架 FasDL，旨在加速无服务器计算范式下深度学习模型的训练速度，减少分布式训练架构下的高通信开销。此外，为了减轻开发者对分布式训练任务的参数配置负担，本文设计了一个自动化的资源参数配置模块，以提升训练框架下模型训练的性能表现。FasDL 训练框架共包含三个主要模块，分别是 K-REDUCE 通信模式、系统性能建模模块和参数配置模块。

K-REDUCE 通信模式通过分析参数聚合阶段函数间通信的特点，选择了最优数量的聚合节点以降低分布式训练期间的通信开销。此外，本文观察到在聚合阶段中非聚合节点处于空闲状态，因此其 CPU 算力没有被充分利用。本文设计了不均等的数据集划分方案和混合异步并行（HAP）协议，以重复利用非聚合节点的空闲算法进一步加速训练。

系统性能建模模块对 FasDL 训练框架下的分布式训练性能进行了建模。本文从端到端训练时间、训练开销和收敛效率三个维度对模型训练的性能进行数学建模。具体地，本文对深度学习模型的训练性能进行分析，并对基于外部存储服务构建的通信通道的性能表现进行了建模。本文设计了一个负载分析模块以对确定分布式训练中训练模型和平台的特性进行详细分析，从而支撑系统性能建模在不同参数配置下对训练性能的预测。

参数配置模块在给定的端到端训练时间约束和全局批次大小约束下，为 FasDL 进行最优的参数配置以最小化训练开销。为了实现高效的参数搜索，本文对各系统参数进行了搜索空间剪枝，以确保模型训练的正常运行。此外，本文设计了一个两阶段的启发式搜索算法，将系统参数之间的关联进行解耦，从而减少了参数组合的搜索空间并降低搜索算法的时间复杂度。

为了验证 FasDL 训练框架的性能，本文在 AWS 平台上开展实验从不同角度对其性能进行分析。本文首先对系统性能建模模块的预测准确度和参数配置模块的参数搜索效果进行了实验验证。其中，系统性能建模模块的预测准确度可以达到 94% 以上，而两阶段启发式搜索算法以轻微的系统性能下降换取了搜索效率的显著提升。然后，本文将 K-REDUCE 通信模式与 LambdaML 进行对比。相比与 ScatterReduce 通

信模式，K-REDUCE 通信模式的训练时间提升了 16.8% 并将训练开销降低了 28.3%，同时能够有效地保障训练的收敛效率。最后，本文对比了不同通信通道实现方案的性能，并对比了 FasDL 训练框架与单机训练的性能。经验证，FasDL 可以兼容不同类型的存储服务，从而最优化不同训练负载的训练性能。相比基于参数服务器的通信实现，如 λDNN，FasDL 最多可降低 78.13% 的训练开销。

6.2　研究展望

本文在无服务器计算平台下，开展了分布式训练的高性能框架的研究。本文结合当前无服务器计算平台的特征，设计了 K-REDUCE 通信模式。然而，诸多工作围绕无服务器计算范式的优化和增强展开，为后续的系统性能优化提供了新的思路。本文从两个方面对未来的研究展望进行了总结。

当前主流的无服务器计算平台采用了以函数的内存配额为中心的资源配置模型。具体地，计算函数的算力和通信带宽均与函数的内存配额直接相关。因此，本文在系统性能建模模块和参数配置模块中对上述资源耦合的情况进行了分析。当前已有工作在研究在资源解耦下的无服务器计算平台的实现。因此，未来如何在新的平台特性下迁移 FasDL 训练框架并保证高性能的训练效率是研究方向之一。

此外，现有工作在研究将无服务器计算下的任务部署到 GPU 上以加速任务执行速度。本文提出的训练框架未涉及使用 GPU 算力加速模型的训练过程。当面对更加复杂的训练负载时，训练框架的训练性能将因此受限。因此，如何在 FasDL 训练框架中借助 GPU 资源进一步加速训练也是未来的研究方向之一。

参考文献

[1] MORENO-VOZMEDIANO R, MONTERO R S, LLORENTE I M. Iaas Cloud Architecture: From Virtualized Datacenters To Federated Cloud Infrastructures[J]. Computer, 2012, 45(12): 65-72.

[2] VAN EYK E, TOADER L, TALLURI S, et al. Serverless Is More: From Paas To Present Cloud Computing[J]. IEEE Internet Computing, 2018, 22(5): 8-17.

[3] TSAI W, BAI X, HUANG Y. Software-as-a-service (SaaS): perspectives and challenges[J]. Science China Information Sciences, 2014, 57: 1-15.

[4] JONAS E, SCHLEIER-SMITH J, SREEKANTI V, et al. Cloud Programming Simplified: A Berkeley View on Serverless Computing[J]. arXiv preprint arXiv:1902.03383, 2019.

[5] LI Z, GUO L, CHENG J, et al. The Serverless Computing Survey: A Technical Primer for Design Architecture[J]. ACM Computing Surveys (CSUR), 2022, 54(10s): 1-34.

[6] SHAFIEI H, KHONSARI A, MOUSAVI P. Serverless Computing: a Survey of Opportunities, Challenges, and Applications[J]. ACM Computing Surveys, 2022, 54(11s): 1-32.

[7] ZHANG M, WANG F, ZHU Y, et al. Serverless Empowered Video Analytics for Ubiquitous Networked Cameras[J]. IEEE Network, 2021, 35(6): 186-193.

[8] ROY R B, PATEL T, TIWARI D. DayDream: Executing Dynamic Scientific Workflows on Serverless Platforms with Hot Starts[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2022: 1-18.

[9] WANG H, NIU D, LI B. Distributed machine learning with a serverless architecture[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. 2019: 1288-1296.

[10] SÁNCHEZ-ARTIGAS M, SARROCA P G. Experience Paper: Towards Enhancing Cost Efficiency in Serverless Machine Learning Training[C]//Proceedings of the International Middleware Conference. 2021: 210-222.

[11] LI M, ANDERSEN D G, PARK J W, et al. Scaling Distributed Machine Learning with the Parameter Server[C]//11th USENIX Symposium on operating systems design and implementation. 2014: 583-598.

[12] HUANG Y, JIN T, WU Y, et al. Flexps: Flexible Parallelism Control in Parameter Server Architecture[J]. Proceedings of the VLDB Endowment, 2018, 11(5): 566-579.

[13] XU F, QIN Y, CHEN L, et al. λdnn: Achieving Predictable Distributed DNN Training with Serverless Architectures[J]. IEEE Transactions on Computers, 2021, 71(2): 450-463.

[14] JIANG J, GAN S, LIU Y, et al. Towards Demystifying Serverless Machine Learning Training[C]//Proceedings of the 2021 International Conference on Management of Data. 2021: 857-871.

[15] KIM J K, HO Q, LEE S, et al. Strads: A Distributed Framework for Scheduled Model Parallel

Machine Learning[C]//Proceedings of the Eleventh European Conference on Computer Systems. 2016: 1-16.

[16] SHOEYBI M, PATWARY M, PURI R, et al. Megatron-lm: Training Multi-billion Parameter Language Models Using Model Parallelism[J]. arXiv preprint arXiv:1909.08053, 2019.

[17] XU A, HUO Z, HUANG H. On the Acceleration of Deep Learning Model Parallelism with Staleness [C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 2088-2097.

[18] DUAN J, QIAN S, YANG D, et al. MOPAR: A Model Partitioning Framework for Deep Learning Inference Services on Serverless Platforms[J]. arXiv preprint arXiv:2404.02445, 2024.

[19] HUANG Y, CHENG Y, BAPNA A, et al. Gpipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism[J]. Advances in neural information processing systems, 2019, 32: 103-112.

[20] PARK J H, YUN G, YI C M, et al. HetPipe: Enabling Large DNN Training on (Whimpy) Heterogeneous GPU Clusters Through Integration of Pipelined Model Parallelism and Data Parallelism [C]//2020 USENIX Annual Technical Conference. 2020: 307-321.

[21] NARAYANAN D, HARLAP A, PHANISHAYEE A, et al. PipeDream: Generalized Pipeline Parallelism for DNN Training[C]//Proceedings of the 27th ACM symposium on operating systems principles. 2019: 1-15.

[22] SHI H, ZHENG W, LIU Z, et al. Automatic Pipeline Parallelism: A Parallel Inference Framework for Deep Learning Applications in 6G Mobile Communication Systems[J]. IEEE Journal on Selected Areas in Communications, 2023, 41(7): 2041-2056.

[23] LIU Y, JIANG B, GUO T, et al. FuncPipe: A Pipelined Serverless Framework for Fast and Cost-Efficient Training of Deep Learning Models[J]. Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2022, 6(3): 1-30.

[24] THORPE J, QIAO Y, EYOLFSON J, et al. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads[C]//15th USENIX Symposium on Operating Systems Design and Implementation. 2021: 495-514.

[25] KHAN T, TIAN W, ZHOU G, et al. Machine learning (ML)-centric resource management in cloud computing: A review and future directions[J]. Journal of Network and Computer Applications, 2022, 204: 103405.

[26] PAN S, ZHAO H, CAI Z, et al. Sustainable Serverless Computing With Cold-Start Optimization and Automatic Workflow Resource Scheduling[J]. IEEE Transactions on Sustainable Computing, 2024, 9(3): 329-340.

[27] JAMIL B, IJAZ H, SHOJAFAR M, et al. Resource allocation and task scheduling in fog computing and internet of everything environments: A taxonomy, review, and future directions[J]. ACM Computing Surveys (CSUR), 2022, 54(11s): 1-38.

[28] FENG L, KUDVA P, DA SILVA D, et al. Exploring Serverless Computing for Neural Network Training[C]//2018 IEEE 11th International Conference on Cloud Computing. 2018: 334-341.

[29] LI Z, LIU Y, GUO L, et al. Faasflow: Enable Efficient Workflow Execution for Function-as-a-service[C]//Proceedings of the 27th acm international conference on architectural support for programming languages and operating systems. 2022: 782-796.

[30] YU M, CAO T, WANG W, et al. Following the Data, not the Function: Rethinking Function Orchestration in Serverless Computing[C]//20th USENIX Symposium on Networked Systems Design and Implementation. 2023: 1489-1504.

[31] KLIMOVIC A, WANG Y, STUEDI P, et al. Pocket: Elastic Ephemeral Storage for Serverless Analytics[C]//13th USENIX Symposium on Operating Systems Design and Implementation. 2018: 427-444.

[32] ZHANG T, XIE D, LI F, et al. Narrowing the Gap Between Serverless and its State with Storage Functions[C]//Proceedings of the ACM Symposium on Cloud Computing. 2019: 1-12.

[33] Amazon Simple Storage Service (S3)[Z]. https://aws.amazon.com/s3/. 2024.

[34] Amazon ElastiCache Serverless[Z]. https://aws.amazon.com/elasticache. 2024.

[35] FOULADI S, ROMERO F, ITER D, et al. From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers[C]//2019 USENIX Annual Technical Conference. 2019: 475-488.

[36] YU T, LIU Q, DU D, et al. Characterizing serverless platforms with serverlessbench[C]//Proceedings of the 11th ACM Symposium on Cloud Computing. 2020: 30-44.

[37] AWS Lambda[Z]. https://aws.amazon.com/lambda/. 2024.

[38] DJEMAME K, PARKER M, DATSEV D. Open-source serverless architectures: an evaluation of apache openwhisk[C]//2020 ieee/acm 13th international conference on utility and cloud computing. 2020: 329-335.

[39] ZHENG H, XU F, CHEN L, et al. Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training[C]//Proceedings of the 48th International Conference on Parallel Processing. 2019: 1-11.

[40] SHANG R, XU F, BAI Z, et al. spotDNN: Provisioning Spot Instances for Predictable Distributed DNN Training in the Cloud[C]//2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS). 2023: 1-10.

[41] CARREIRA J, FONSECA P, TUMANOV A, et al. Cirrus: A serverless framework for end-to-end ml workflows[C]//Proceedings of the ACM Symposium on Cloud Computing. 2019: 13-24.

[42] SAHA A, JINDAL S. EMARS: Efficient Management and Allocation of Resources in Serverless [C]//2018 IEEE 11th International Conference on Cloud Computing. 2018: 827-830.

[43] CHEN J, XU F, GU Y, et al. HarmonyBatch: Batching multi-SLO DNN Inference with Heterogeneous Serverless Functions[J]. arXiv preprint arXiv:2405.05633, 2024.

[44] HU H, LIU F, PEI Q, et al. Grapher: A Resource-Efficient Serverless System for GNN Serving through Graph Sharing[C]//The Web Conference 2024. 2024: 2826-2835.

[45] PEI Q, YUAN Y, HU H, et al. Asyfunc: A high-performance and resource-efficient serverless inference system via asymmetric functions[C]//Proceedings of the 2023 ACM Symposium on Cloud Computing. 2023: 324-340.

[46] LIU F, NIU Y. Demystifying the Cost of Serverless Computing: Towards a Win-Win Deal[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(1): 59-72.

[47] WEN Z, CHEN Q, NIU Y, et al. Joint Optimization of Parallelism and Resource Configuration for Serverless Function Steps[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(4): 560-576.

[48] LIN C, KHAZAEI H. Modeling and optimization of performance and cost of serverless applications [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(3): 615-632.

[49] LIAN X, ZHANG C, ZHANG H, et al. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent[J]. Advances in neural information processing systems, 2017, 30: 5331-5341.

[50] MAMPAGE A, KARUNASEKERA S, BUYYA R. Deep reinforcement learning for application scheduling in resource-constrained, multi-tenant serverless computing environments[J]. Future Generation Computer Systems, 2023, 143: 277-292.

[51] HE K, ZHANG X, REN S, et al. Deep Residual Learning for Image Recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[52] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[53] BILAL M, CANINI M, FONSECA R, et al. With Great Freedom Comes Great Opportunity: Rethinking Resource Allocation for Serverless Functions[C]//Proceedings of the Eighteenth European Conference on Computer Systems. 2023: 381-397.

[54] YU H, WANG H, LI J, et al. Accelerating Serverless Computing by Harvesting Idle Resources[C]//Proceedings of the ACM Web Conference. 2022: 1741-1751.

[55] Amazon Web Services. AWS Lambda – FAQ[Z]. https://aws.amazon.com/lambda/faqs/. [Online; accessed 13-March-2024]. 2024.

[56] MAHGOUB A, YI E B, SHANKAR K, et al. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs[C]//16th USENIX Symposium on Operating Systems Design and Implementation. 2022: 303-320.

[57] GERBESSIOTIS A, VALIANT L. Direct Bulk-Synchronous Parallel Algorithms[J]. Journal of Parallel and Distributed Computing, 1994, 22(2): 251-267.

[58] BRADLEY J K, KYROLA A, BICKSON D, et al. Parallel Coordinate Descent for L1-regularized Loss Minimization[J]. arXiv preprint arXiv:1105.5379, 2011.

[59] HO Q, CIPAR J, CUI H, et al. More Effective Distributed ML Via a Stale Synchronous Parallel Parameter Server[J]. Advances in Neural Information Processing Systems, 2013, 26: 1223-1231.

[60] BOYD S, BOYD S P, VANDENBERGHE L. Convex Optimization[M]. 2004.

[61] HOWARD A G. Mobilenets: Efficient convolutional neural networks for mobile vision applications [J]. arXiv preprint arXiv:1704.04861, 2017.

[62] DEVLIN J, CHANG M W, LEE K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

[63] KRIZHEVSKY A. Learning Multiple Layers of Features from Tiny Images[R]. 2009.

[64] WARSTADT A, et al. The Corpus of Linguistic Acceptability (CoLA)[Z]. https://nyu-mll.github.io/cola/. 2018.

致 谢

在上海交通大学电子信息与电气工程学院，我度过了充实的两年半硕士学习生活。我在这里完成了研究生期间的专业课程学习，提升了我个人的学术和技术水平。同时，我也认识了很多友善且有趣的朋友、同学和老师。这两年半的学习生活将会成为我未来工作和生活的珍贵回忆。

感谢我的导师马汝辉教授，在整个研究生学习期间对我的悉心指导和关怀。在学术研究的道路上，每当我陷入迷茫，是导师凭借其深厚的专业知识和敏锐的学术洞察力，为我拨开迷雾。导师也非常注重对我的个人成长和职业发展的引导。鼓励我积极参加各类项目活动，提升技术水平，培养独立思考和解决问题的能力。祝愿马老师工作顺遂，生活美满。

感谢蔡子诺学长以及课题组的同门伙伴们，在我的研究生期间与我的相处与交流。在学术交流上，我们可以互相交流问题，共同探索潜在的研究方向。在日常生活中，我们可以在课题组团建时畅所欲言，一起讨论研究生生活的趣事。祝愿大家在未来都能科研顺利，前程光明。

感谢我的父亲、母亲、妹妹以及亲人们对我的支持和帮助。你们始终作为我最坚强的后盾，鼓励和支持着我在求学道路上不断求索。你们的帮助和关怀我会时刻铭记。

感谢我的朋友们在两年半里相处和陪伴，你们是我学习和科研生活之余的快乐源泉。我们一起在交大校园里留下了许多美好的回忆和欢笑。希望这份情谊能够继续延续至我们未来的生活中。

学术论文和科研成果目录

学术论文

[1] X. Chen, Z. Cai, h. Zhang, r. ma and R. Buyya, "FasDL: An Efficient Serverless-Based Training Architecture with Communication Optimization and Resource Configuration," in IEEE Transactions on Computers, vol. 74, no. 2, pp. 468-482, Feb. 2025, doi: 10.1109/TC.2024.3485202.

[2] Z. Cai, Z. Chen, X. Chen, R. Ma, H. Guan and R. Buyya, "SPSC: Stream Processing Framework Atop Serverless Computing for Industrial Big Data," in IEEE Transactions on Cybernetics, vol. 54, no. 11, pp. 6509-6517, Nov. 2024, doi: 10.1109/TCYB.2024.3407886.

专利

[3] 陈星耒, 基于无服务器架构的高效分布式机器学习训练系统, 专利申请号202411471734.8.