基于无服务器 MapReduce 的词频统计

Hanzhe Li[1], Bingchen Lin[2], Mengyuan Xu[3]

[1] 西安交通大学人工智能学院，陕西省西安市咸宁西路，中国 [2] 重庆第二师范学院人工智能学院，重庆市崇教路，中国 [3] 齐鲁理工学院计算机与信息工程学院，山东省济南市经十东路，中国 [1]*lhz2023@stu.xjtu.edu.cn*, [2]*2311401218@stu.cque.edu.cn*, [3]*20202270@stu.hebmu.edu.cn*

关键词: 无服务器计算；MapReduce；词频统计；云计算

摘要: 随着对高性能、高效率计算需求的不断增长，云计算，尤其是无服务器计算，近年来逐渐成为研究热点，吸引了众多研究者的关注。与此同时，作为业界广泛采用的大数据处理模型，MapReduce 已应用于各个领域。受函数即服务的无服务器框架以及 MapReduce 编程模型的高并发性和鲁棒性启发，本文重点探讨将二者结合，以缩短执行词频统计任务所需的时间并提高效率。为此，本文采用基于无服务器计算平台的 MapReduce 编程模型，确定针对特定任务的最优 Map 函数和 Reduce 函数数量。对于相同的工作负载，大量实验表明，随着 Map 函数和 Reduce 函数数量的增加，执行时间会以不同幅度缩短，程序的整体效率也会提高。本文认为，找到最优的 Map 函数和 Reduce 函数数量，有助于企业和程序员确定最优解决方案。

1 引言

为满足实际应用中对计算资源和高端芯片组日益增长的需求（McGrath 等人，2017年；Baldini 等人，2017年），近年来，云计算技术吸引了越来越多的研究关注，并形成了多种经典的云服务模式，例如基础设施即服务（IaaS）、平台即服务（PaaS）和软件即服务（SaaS）。然而，上述模式依赖较高水平的专业知识，成本高昂，且无法在管理、扩展和成本效益指标之间实现平衡。为缓解上述问题，无服务器计算应运而生，旨在减轻服务器管理负担并节省云服务成本（Vincent 等人，2019年）。

无服务器计算的基本单元是函数。收到用户请求后，无服务器平台会根据请求中的参数（例如函数的 URL），调用平台上的相关函数。这种服务模式通常称为函数即服务（FaaS），通常与后端即服务（BaaS）配合使用。与传统的集中式单体应用相比，FaaS 服务由明确编排的独立函数组成，能够直观地呈现应用的业务逻辑控制和数据流。此外，无服务器计算更加经济实惠，因为用户不再需要为闲置的额外计算资源、已使用资源的维护以及这些资源的安全性付费。

https://orcid.org/0009-0002-8999-7996
https://orcid.org/0009-0001-8866-7752
https://orcid.org/0009-0005-0411-3656

无服务器计算使用户能够更加专注于程序逻辑。至于后端服务器的维护，则完全由服务提供商负责。（McGrath 等人，2017年；Jeffrey 等人，2004年）与传统的本地计算服务器相比，无服务器计算具有更强的可扩展性和弹性，因为计算资源的动态分配使用户能够应对工作负载和数据处理需求的突然激增。目前，许多无服务器计算平台都提供先进的云计算服务，例如 AWS Lambda、Google Cloud、Microsoft Azure、阿里云等。

MapReduce 是目前处理海量数据最流行的模型，主要包含四个阶段：Map、Partition、Shuffle 和 Reduce。MapReduce 广泛应用于分布式系统中的并行处理以及大规模数据集的生成。首先，它易于使用，即使是初学者也能轻松上手，因为它隐藏了并行化、容错、优化局部性和均衡工作负载等方面的具体复杂性。其次，现实世界中的许多复杂问题都能很好地用 MapReduce 编程模型来表达，例如词语计数、词频分析等（Baldini 等人，2017）。然而，MapReduce 往往受数据传输方式的限制。具体而言，由于映射器需要尽快完成，当归约器仍在工作时，映射器可能面临超时风险。因此，无法直接在映射器和归约器之间传输数据。在此背景下，结合无服务器架构与 MapReduce 框架展现出广阔的应用前景。

受这两项前沿且成熟的技术启发，本文着眼于将它们结合起来，以缩短执行词频统计任务所需的时间并提高效率。本文基于无服务器计算平台，采用 MapReduce 编程模型，旨在确定 Map 函数和 Reduce 函数的最优数量。尽管乍看之下，似乎实现的 Map 函数和 Reduce 函数越多，程序的整体效率就越高，但本文的目标是探究整体效率提升的趋势。结果表明，在处理相同工作负载时，随着 Map 函数和 Reduce 函数数量的增加，执行时间都会缩短，程序的整体效率也会提高，但两者的变化速率不同。本文希望找出 Map 函数和 Reduce 函数的最优数量，从而帮助企业和程序员确定最优解决方案以便在各自的任务和工作流中实现 MapReduce 编程模型。

围绕上述方面，本文首先简要概述 MapReduce 编程模型的基本原理、无服务器计算平台的运行规则与服务，以及实验的整体框架（第2节）。随后，本文探讨相关方法与评估，并基于现有研究数据和结果，通过深入评估与分析得出实验结果和结论（第3节）。最后，本文讨论所用实验框架目前存在的不足，分析结果的优势与劣势，并基于当前实验展望可能的解决方案和新的研究领域（第4节）。本文还在第4节进行了总结。

2 方法

2.1 重新审视 MapReduce 与无服务器计算

本节简要概述 MapReduce 编程模型的基本原理，以及无服务器计算平台的运行规则。

MapReduce。MapReduce 编程模型总体上主要由两个函数、两个阶段和三类文件构成。三类文件分别是输入文件、中间文件和输出文件。输入文件包含待处理的数据。中间文件包含 MapReduce 执行过程中所需的重要数据，而输出文件则保存程序的最终结果。就两个函数和两个阶段而言，Map 函数对应 Map 阶段，Reduce 函数对应 Reduce 阶段。Map 函数负责从输入文件中读取数据，并将这些数据处理为键值对，随后存储在中间文件中。这些中间文件将键值对传递给 Reduce 函数，由后者对键值对进行排序、分区和处理，生成最终结果，并将其写入输出文件，供用户获取和访问（Jeffrey 等人，2004年）。

无服务器。无服务器计算平台的运行规则由四个主要阶段组成，分别是：事件触发、函数执行、函数处理和响应返回。在事件触发阶段，有一个本地客户端运行在用户的设备上。客户端会触发一个事件，例如 HTTP 请求、文件上传或

消息队列。随后，触发器将事件传递给云端的函数，进入函数执行阶段。云端函数被触发后，无服务器计算平台将动态分配并扩展计算资源，以开始执行函数。在函数处理阶段，函数中的代码会被执行并生成输出。在此过程中，平台上的函数将获得授权，以访问和操作用户预先请求的存储、数据库及其他相关服务。最后，在响应返回阶段，函数将输出返回给用户的本地客户端。响应可以是任何形式的内容，例如响应数据、状态更新以及各种形式的通知等。通常，函数生成的结果会存储在无服务器计算平台提供的存储服务中。

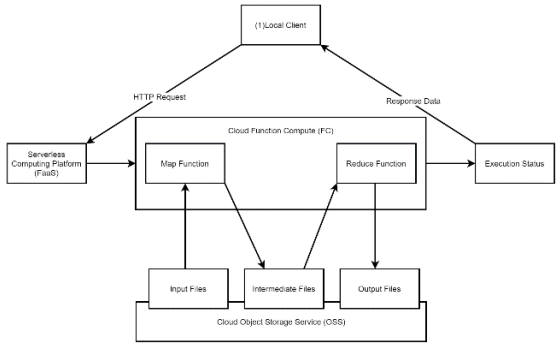


图1：所提出方法的框架。

2.2 总体框架

本文的主要目标是基于无服务器计算，利用 MapReduce 进行词频分析。为将 MapReduce 编程模型、无服务器计算和词频分析完美结合，本文实现了以下方法，并对 MapReduce 编程模型进行了少量修改。

整个实验首先采用控制变量法进行。本文对同一组文本文档进行分析，这些文档采用文本文档格式，但使用参数不同的 MapReduce 框架。参数差异涉及 MapReduce 函数的数量、无服务器网络上 CPU 和内存的配置等。因此，在整个实验过程中，可以通过对这些参数进行调整来分析性能变化。

其次，以下介绍整个实验过程中使用的设备。如图1所示，本文的无服务器 MapReduce 框架包含一个部署在 PyCharm 中的本地 Python 客户端。该客户端负责调用部署在无服务器平台上的函数，并在 MapReduce 函数成功执行后接收完成信号。实验中使用的无服务器计算平台是阿里云平台。本文具体使用的服务包括阿里云函数计算（FC），团队在其中部署 MapReduce 函数；以及阿里云对象存储服务（OSS），团队在其中临时存储与本实验相关的文件，以便所有需要读写文件的操作都在无服务器平台上进行，确保本地与云端之间的数据传输速度不会显著影响执行时间。

最后，本文所使用的待分析文件质量相同。每个文件大约包含 1,000,000 个词。确保这些文件的词数大致相同至关重要，因为不同的工作负载也会导致各项测试的性能出现差异。

2.3 MapReduce 函数

对于 MapReduce 函数的细微改动，本文自定义了 Map 函数读取数据的方式。本实验中的每个 Map 函数都包含一组参数，分别为“文件ID”、“文件数量”和“索引”。这些参数可帮助 Map 函数读取存储在阿里云 OSS 中的正确文件组，从而确保每个文件在执行过程中只被处理一次。

3 实验

3.1 实验设置

在现代技术领域，提升计算速度始终是研究人员和技术工程师关注的重点（Zhenyu等人，2023年）。为了实现更高效的计算，本文在本次实验中采用了一种新策略（Jeffrey 等人，2004 年），即用多线程技术取代单线程，以优化计算速度。简单来说，多线程技术

是指同时执行多个任务，从而在同一时间完成更多工作。与单线程相比，多线程能够在相对较短的时间内完成更多任务，从而提升整体计算速度。这项技术在计算机科学领域取得了显著成果，尤其是在处理大型复杂数学模型、大规模数据分析和实时通信方面，具有明显优势。

由于本文旨在探讨如何通过应用多线程技术提升计算速度来实现无服务器 MapReduce，团队首先分析了现有的单线程程序，并确定了后续多线程设计中需要优化的瓶颈部分。随后，团队设计了相应的多线程算法，并对其进行了详细分析和测试。在整个实验过程中，团队不断调整和改进多线程策略，以尽可能显著地提升计算速度。电脑本地有一个名为 client 的编码文件。团队在控制台中找到 FC，并在其函数服务中创建了一个函数。在函数服务中，团队编写了 Map 函数和 Reduce 函数。

实验中使用了 50 个文件，每个文件大约包含 1,000,000 个单词。团队首先使用本地的一段代码和网络协议，将这些文件上传到阿里云对象存储服务（OSS）。随后，团队提前在阿里云函数计算（FC）中创建了 Mapper 函数和 Reducer 函数。平台上的每个函数均部署在 0.35 个虚拟 CPU 上，并配置了 512MB 内存。之后，团队在本地启用了预先准备好的客户端代码，将存储在 OSS 中的 50 个文件调用到阿里云函数计算（FC）中，从而启动程序运行。

以往研究（J Jiang 等，2021；Prasoon 等，2024）表明，可以根据 MapReduce 编程模型和无服务器计算的执行时间跨度及内存使用量来评估其性能。一方面，执行时间是程序性能的直接反映。另一方面，内存使用量体现了执行期间的资源管理与分配情况，使团队能够更清晰地观察结果。此外，团队能够优化分配给每个函数的工作负载，并同步改进算法，从而在整个实验过程中改进方法（Rodrigo 等，2024；Q Liu 等，2024）。

表 1：不同数量 MapReduce 函数下的模型性能比较（使用 50 个文件）

Func Num	平均执行时间 /毫秒		平均内存使用量 /MB	
	映射器	归约器	映射器	归约器
1	40816.58	51624.64	1604.26	1021.02
2	7716.69	15133.26	821.54	533.82
5	2455.69	4269.94	351.06	246.82
10	1464.97	2198.27	194.01	139.54

3.2 不同数量 MapReduce 函数的 RAM 使用量

团队首先定量比较了不同 MapReduce 函数对 RAM 使用量的影响，结果如表 1 所示。在实验的第一种情况下，团队仅使用一个 MapReduce 函数。映射器函数的平均执行时间为 40816.58ms，归约器函数的平均执行时间为 51624.64ms，映射器和归约器使用的 RAM 分别为 1021.02 MB 和 1604.26 MB。在实验接下来的情况下，部署了两组 MapReduce 函数。映射器函数的平均执行时间为 7716.69ms，归约器函数的平均执行时间为 15133.26ms。映射器和 Reduce 使用的 RAM 分别为 533.82MB 和 821.54MB。在实验的第三种情况下，团队使用五个 MapReduce 函数。映射器函数的平均执行时间为 2455.694ms，归约器函数的平均执行时间达到 4269.94ms。映射器和归约器使用的 RAM 分别为 246.824MB 和 351.066MB。在实验的最后一种情况下，团队使用 10 个 MapReduce 函数。映射器函数的平均执行时间为 1464.974ms，归约器函数的平均执行时间达到 2198.27ms。映射器和归约器使用的 RAM 分别为 139.545MB 和 194.016MB。

3.3 不同数量 MapReduce 函数的时间开销

如图 2 所示，结果表明，随着 MapReduce 函数数量的增加，平均执行时间逐渐缩短。这表明，在大数据处理过程中，合理增加 MapReduce 函数的数量，可以有效提高

数据处理效率并缩短执行时间 （J Cai 等人，2023年）。然而，这些结果也表明，盲目增加 MapReduce 函数的数量并非有效方法，因为执行时间缩短的幅度正在下降。因此，团队得出一个简要结论：配置 MapReduce 函数数量时，最好根据工作负载和现有资源进行调整，这样既能尽可能获得理想的结果，又不会消耗过多资源或造成过高成本。

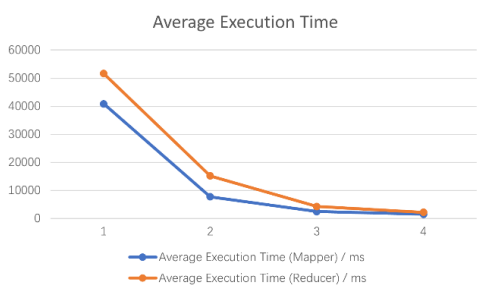


图 2：MapReduce 函数的平均执行时间。

3.4 内存使用量与平均使用时间对比

完成所有实验工作后，团队重点关注了 MapReduce 函数运行时的内存使用量。如图 3 所示，随着 MapReduce 函数数量的增加，平均内存使用量也呈下降趋势。这可能是因为函数数量增加后，系统可以在更多核心上并行执行任务，从而减少单个任务的内存占用。此外，通过优化 MapReduce 函数的编写方式和执行策略，团队还能进一步降低内存使用量并提高系统资源利用率。

团队分析了每次实验中映射器和归约器的平均使用时间占比。通过比较，团队发现，不能简单地假设随着 MapReduce 函数数量增加，映射器处理数据所耗费的时间就会变长。在实验过程中，团队使用不同数量的 MapReduce 函数，并对每次实验进行了详细的计时分析。结果表明，不同实验中映射器和归约器的使用时间占比存在一定差异。这说明，MapReduce 函数数量增加并不一定会缩短映射器的处理时间。

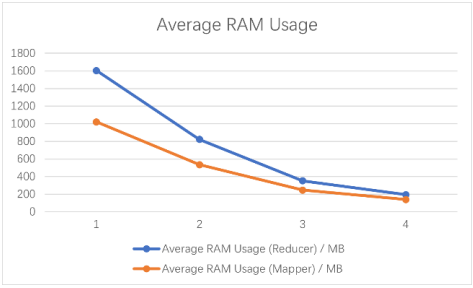


图 3：函数的平均 RAM 使用量。

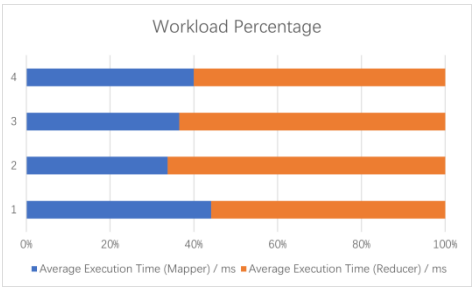


图 4：MapReduce 函数的工作负载百分比。

3.5 MapReduce 实现的影响分析

如上方图 4 所示，团队还探究了 MapReduce 函数的内部实现对映射器和 Reduce 使用时间占比的影响。通过比较不同实验中的代码实现，团队发现，优化 MapReduce 函数的内部实现可能会改变映射器和 Reduce 的使用时间占比。这意味着，在增加 MapReduce 函数数量时，优化其内部实现可以有效缩短映射器的处理时间，从而提高整体计算效率。为此，要提升整体计算性能，团队需要关注数据大小、MapReduce 函数的内部实现及其他影响因素，以便在实际应用中实现更高效的分布式计算。

4 结论与未来工作

本文介绍了一种基于无服务器计算平台的多线程 MapReduce 框架，旨在提高程序的整体效率，同时借助其易用性，最大限度地减少本地服务器维护工作。

无服务器计算平台。当使用基于无服务器平台的 MapReduce 分析词频时，团队首先发现，随着 MapReduce 函数（也称为线程）数量的增加，程序的执行速度也会随之提升。然而，速度提升的幅度存在峰值，这意味着盲目增加 MapReduce 函数的数量可能会浪费计算资源，或降低无服务器计算资源的利用效率。

为此，本文围绕 MapReduce 函数的数量开展了一系列无服务器 MapReduce 实验，并根据执行过程中的平均执行时间和平均内存使用量评估结果。研究团队推测，执行时间的下降幅度会先显著增大，随后逐渐放缓。因此，团队得出结论：盲目增加 MapReduce 函数的数量并不总能提高程序效率；对于不同任务，应分别谨慎地确定 MapReduce 函数的数量，以充分利用计算资源（Prasoon 等，2024）。

研究团队还注意到，将这项工作拓展至实际应用时存在一些局限性，主要源于一项理想化设定：每个词频文件的工作负载大致相同。此外，团队开展的词频分析任务并非普遍适用，因为该任务对计算资源的需求较低。因此，还需要研究更多类型的任务，以完善这项研究。团队未来的工作包括探索如何根据不同工作负载动态分配 MapReduce 函数，从而为每个词频文件配置足够数量的 MapReduce 函数，以便在执行分布于不同文件且负载不均衡的任务时实现更高效率。

作者贡献

所有作者贡献相同，姓名按字母顺序排列。

参考文献

McGrath, G., Brenner, P. R., 2017年。无服务器计算：设计、实现与性能。发表于 *2017*年 *IEEE* 第 *37* 届分布式计算系统国际会议研讨会，405-410。Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., 等，2017年。无服务器计算：当前趋势与开放问题。载于《云计算研究进展》，1-20。Vincent, A., Germán, M., Miguel, C., 2019年。基于 AWS Lambda 的无服务器 MapReduce 框架与性能评估。*Future Generation Computer Systems,* 97（2019年）259–274。Jeffrey, D., Sanjay, G., 2004年。MapReduce：大型集群上的简化数据处理。载于 *OSDI'04*：第 *6* 届操作系统设计与实现研讨会。USENIX 协会。Zhenyu, Y., Tianyang, N., Min, L., 2023年。混合存储模式下的 MapReduce 作业调度。J Jiang, S Gan, Y Liu, F Wang, Gustavo A., Ana K., Ankit S., W, Wu, 和 C Zhang，2021年。迈向揭开无服务器机器学习训练的奥秘。载于 *2021* 年数据管理国际会议论文集（*SIGMOD'21*）。计算机协会，纽约，美国，857-871。Prasoon S., Kostis K., Neeraja J. Y., 2024年。Shabari：通过延迟决策实现更快速、高效的无服务器函数。*arXiv* 预印本 *arXiv:2401.08859*。Rodrigo L. A., Mahdi Z., 2024 年。无限 FaaS：通过调用驱动架构和内存检查点突破无服务器函数的执行时间限制。载于 *arXiv:2402.09377*。Q Liu, Y Yang, D Du, Y Xia, P Zhang, J Feng, James, L., H Chen, 2024年。Jiagu：以协调兼顾效率与实用性来优化无服务器计算资源利用率。载于 *arXiv:2403.00433v1*。J Cai, K Huang, Z Liao，2023年。效率评估 o f 无服务器平台上的MapReduce算法，载于*2023*年 *IEEE*第三届电子技术、通信与信息国际会议（*ICETCI*），中国长春。