中国科学院大学
University of Chinese Academy of Sciences

研究生学位论文开题报告

报告题目	事件驱动的 FaaS 编排执行系统设计与实现		
学生姓名	高浩城	学号	2021E8015082055
指导教师	陈 伟	职称	副研究员
学位类别	工学硕士		
学科专业	软件工程		
研究方向	分布式软件理论与技术		
培养单位	中国科学院软件研究所		
填表日期	2023 年 6 月 29 日		

中国科学院大学制

填 表 说 明

1. 本表内容须真实、完整、准确。
2. “学位类别”名称填写：哲学博士、教育学博士、理学博士、工学博士、农学博士、医学博士、管理学博士，哲学硕士、经济学硕士、法学硕士、教育学硕士、文学硕士、理学硕士、工学硕士、农学硕士、医学硕士、管理学硕士等。
3. “学科专业”名称填写： “二级学科”全称。

摘要

随着云计算和无服务器计算的发展，函数即服务（FaaS）正在改变业务逻辑的实现方式，其在数据分析、优化算法等并行计算任务中有巨大的潜力，同时也带来了新的挑战，尤其是在 FaaS 任务的编排和执行问题上，存在着调度实验高、执行效率低、难以支持复杂结构等问题。针对这些问题，本工作将研究和实现一种基于事件驱动模型的 FaaS 编排执行系统。论文工作将首先定义一种基于“触发器-动作”的任务编排模型，以增强处理复杂执行场景的灵活性。其次，将实现一种事件驱动的函数调度执行机制，该机制允许在特定条件满足的情况下动态地触发函数，并允许函数重叠执行（overlapping execution），即允许多个函数在一定时间内并行执行。在系统实现方面，论文工作将基于发布/订阅（pub/sub）机制与消息队列系统，实现事件驱动的 FaaS 任务执行，以优化调度开销。最后，将构建一个基于事件驱动模型的 FaaS 编排执行系统，包含任务调度器、任务管理器、资源管理器、事件处理器等关键组件，提供更灵活、可扩展的 FaaS 编排执行方式。最后，本工作将通过实验验证方法在提高执行效率、降低调度时延和支持复杂任务结构等方面的有效性。

目录

1. 选题的背景与意义

1.1 无服务器计算与 FaaS 任务编排

随着云计算的普及和无服务器计算[1]的发展，业务逻辑的实现方式正发生着深刻的变化。函数即服务（Function as a Service, FaaS）是一种计算模型，它将云计算资源的抽象提升至函数（function）粒度。用户在 FaaS 环境中，只需将业务逻辑打包成一个或者多个函数，然后上传到 FaaS 平台，平台会自动根据需要进行函数的调度和执行。此外，FaaS 也遵循“按需付费”模式，即只有函数在执行的时候用户才需要付费，而不使用的时候不产生费用。在 FaaS 模式下，开发者只需要关注业务逻辑的实现，而无需管理服务器或者运行环境，大大降低了开发和运维的复杂性。

在函数即服务（FaaS）中，每个函数通常被设计成执行一个特定的、较为细化的任务。当处理更复杂的任务或实现更为复杂的业务逻辑时，通常需要将多个这样的函数组合以完成整体的任务。主要实现此类函数组合的两种方式为协同（Choreography）和编排（Orchestration），各自具有不同的工作方式和特性。

协同基于去中心化模式，不存在一个中心协调器或调度器，每个任务都是自主的，他们通过定义好的规则和协议进行交互和协作。协同具有松散耦合，允许动态交互，系统弹性高等特点，在处理大规模并行任务时，可以很好的发挥优势。然而，缺点也很明显，由于没有统一的控制，当业务逻辑复杂或者需要全局视图和控制时，协同方式会变得非常复杂和困难。

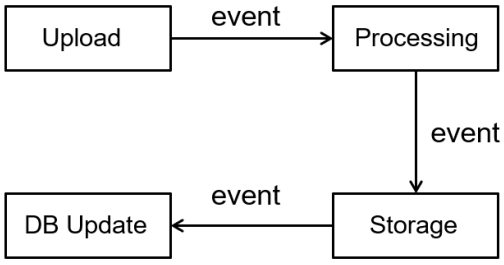

图 1-1 协同模式的函数服务调用示例

编排则是一种集中式的解决方案，由中央编排器，负责调度和管理所有的任

务，确定他们的执行顺序和依赖关系。编排器可以拥有全局的视图和控制，确保所有的任务都能按照预期的方式进行。编排的优势在于它的可控性和可管理性，对于复杂的业务逻辑和流程，编排方式更为直观和易于理解和管理。然而，编排模式会引入额外的调度时延，包括网络延迟和中心节点的处理延迟，这可能对系统性能产生影响，并且当任务规模非常大时，中心编排器的压力也会很大。

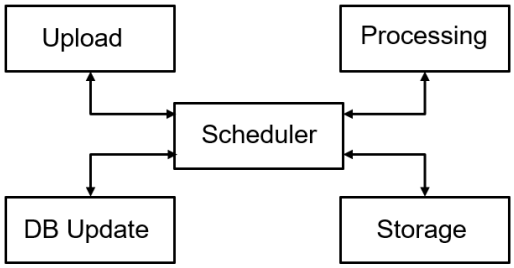

图 1-2 编排模式的函数服务调用示例

鉴于编排模式提供了明确且易于管理的业务逻辑，将函数以工作流的形式进行编排已经成为目前的主流做法，众多的商业云服务提供商和开源平台均提供了此类编排模式的服务。然而，进一步优化其性能和可用性，降低系统延迟和资源消耗，以更好地适应复杂、大规模的实时应用，仍然是一个具有挑战性的研究问题。

1.2 事件驱动模型

鉴于上述挑战，将事件驱动模型引入编排模式成为一种具有潜力的解决方案。事件驱动模型是一种以事件为核心的程序设计范式，其本质是在接收到某个事件（如用户输入、系统消息等）时启动相应的代码来进行响应。这种模型突破了传统的预定义执行路径，将控制流程的焦点集中在事件的产生与处理上。事件驱动模型在解决 FaaS 编排与执行问题上具有以下显著的优势：

低调度时延。中央调度器仅负责 FaaS 任务的划分与放置，而函数运行时的调度管理则由各计算节点负责。这种设计显著减轻了控制节点的负载压力，从而有效地降低了调度时延。

高灵活性与可伸缩性。在事件驱动模型中，函数之间的依赖关系得以解耦，这为编排更加复杂且灵活的 FaaS 任务提供了可能，并且在事件驱动的环境中，

各个组件可以独立地响应和处理事件，从而在面对大规模或者高变化的工作负载时，实现更高效的资源利用。

统一的控制流和数据流。传统的 FaaS 编排执行系统通常基于控制流，而在事件驱动模型中，本工作试图将控制流与数据流合二为一，为用户提供一个统一的、高效且直观的编程模型，以满足更多元化的计算需求，提升系统的整体效率。

支持复杂结构。通过事件驱动架构和度数据流的细粒度管理，FaaS 编排系统可以表现出更强的表达力，以满足更多样化、更复杂的任务需求。函数可以根据各自接收到的事件触发，并进行独立的处理和响应，从而构成非线性、高度交互的复杂任务结构。

基于上述背景与考虑，本工作选择事件驱动模型作为 FaaS 编排执行系统的基础模型。期望通过其高度的灵活性、可扩展性，以及其对复杂系统的管理能力，来推动 FaaS 在复杂业务场景中的广泛应用。

2. 国内外研究现状与趋势

目前关于 FaaS 编排与执行和事件驱动模型，业界与学术界均已有了一定的研究成果。本节将从 FaaS 编排与执行、弱中心化的 FaaS 编排策略、数据驱动的函数编排与优化、事件驱动系统在任务编排中的应用四个方面来介绍相关的研究现状与发展趋势。

2.1 FaaS 编排与执行

目前各大云提供商均提供了 FaaS 任务编排产品，如 AWS Step Functions[3]、Microsoft Durable Functions[10]、Google Workflows[4]、阿里巴巴 Serverless 工作流[11]等。开源的无服务器系统 OpenWhisk[2]和 fission[12]也支持处理序列工作流。

当前的无服务器平台采用以函数为中心的方法来编排和激活无服务器函数，以工作流的形式管理 FaaS 函数序列：每个函数都被视为一个单独的、独立的单元，函数之间的交互在工作流内单独表示。该工作流根据函数的调用依赖性连接各个函数，使得每个函数可以在一个或多个上游函数完成后被触发。例如，许多平台将无服务器任务模型化为有向无环图（DAG）[2, 3, 4, 5, 6, 7, 8, 9]，其中节点代表函数，边表示函数之间的调用依赖关系。可以使用通用编程语言[2][4]或特定领域的语言如 Amazon States Language[3][5]来指定 DAG。

计算和数据密集型应用的某些类别具有并行性质，可被构建为由短的、细粒度任务组成的有向无环图（DAG）。无服务器平台的大规模并行性和自动扩展服务特性，对这种由众多突发并行细粒度任务构成的基于 DAG 的工作流极其适用。这类并行应用包括数据分析[27]、优化算法[28]，以及实时机器学习分类（如支持向量机，SVM）[29][30][31]，它们常常在大规模并行环境下需求低延迟调度。

函数即服务（FaaS）以精细的颗粒度计费函数执行时间，例如，AWS Lambda 按调用次数计费。这样的计费模式使得短任务工作负载能有效利用精细的按使用付费定价模型以降低成本。因此，无服务器计算可以被视为下一代突发并行工作负载在高性能计算（HPC）和数据分析方面的潜在利器。

将此类应用从传统的服务器部署迁移到无服务器平台带来了独特的机会。传统服务器部署依赖于像 MapReduce[32]、Apache Spark[33]、Sparrow[34]和 Dask[35]这样的任务管理框架提供逻辑上的集中调度器，以管理任务和资源分配。这些调度

器通常是在预设定的、可知的硬件资源上运行，他们的资源（如 CPU、内存和存储）在调度任务时必须被显式管理和调度。然而，无服务器计算的特性使得不再需要这样的调度器，原因在于：（1）FaaS 提供商承担了管理“服务器”（即任务执行器的托管地）的责任；（2）资源按照函数的调用次数或执行时间计费使用。在无服务器计算中，开发者无需管理基础设施，只需要关心业务逻辑，然后由平台负责管理和扩展资源。

然而，面向无服务器的计算框架面临着新的的挑战。首先，尽管无服务器平台（如 AWS Lambda）承诺提供超强的弹性和自动扩展属性，但无服务器调用模型会带来不小的调度开销，例如函数启动时延、调度延迟等，并且虽然 FaaS 提供商负责资源管理和负载均衡，但在设计无服务器计算框架时，也需要考虑如何有效利用资源，这关乎到计算效率和计费模式。这说明简单地尝试将现有的服务器 DAG 框架移植到无服务器计算将是不成功的，研究和实现对 FaaS 任务的精细划分和编排成为了当前 FaaS 研究的一个重要问题。

2.2 弱中心化的 FaaS 编排策略

尽管中心化的 FaaS 任务编排有其优势，但是存在一些挑战。例如，网络延迟：在中心化的架构中，所有的任务调度请求都需要通过控制节点，这可能引入额外的网络延迟。负载管理：所有的调度请求将集中于控制节点，随着 FaaS 任务数量的增加，可能导致该节点成为性能瓶颈，并且其故障可能影响整个系统的运行。扩展性问题：随着系统规模的增长，控制节点需要处理越来越多的调度请求，这对其扩展性提出了严峻的挑战。因此，一些研究尝试将部分调度职责分配给计算节点，通过采用弱中心化的策略来缓解控制节点的负载压力。

FaaSFlow[13]提出了 WorkerSP (Worker-side workflow Schedule Pattern)，不同于现有的集中式的 MasterSP (Master-side workflow Schedule Pattern) 架构，它将触发的函数功能随机分配给 Worker 节点以实现负载平衡，FaaSFlow 为了实现 WorkerSP 架构，在 Worker 节点上提出了共享主存，将一个函数功能固定放置在一个节点进行管理。相较于 MasterSP，WorkerSP 架构中，Master 节点仅仅负责对 FaaS 任务的划分放置进行迭代，Worker 节点负责任务整个过程的调度与执行，因为 Worker 节点会存放函数之间的数据和控制依赖信息以及 DAG 划分放置的信息，所以 Worker 可以自行对任务进行调度执行。

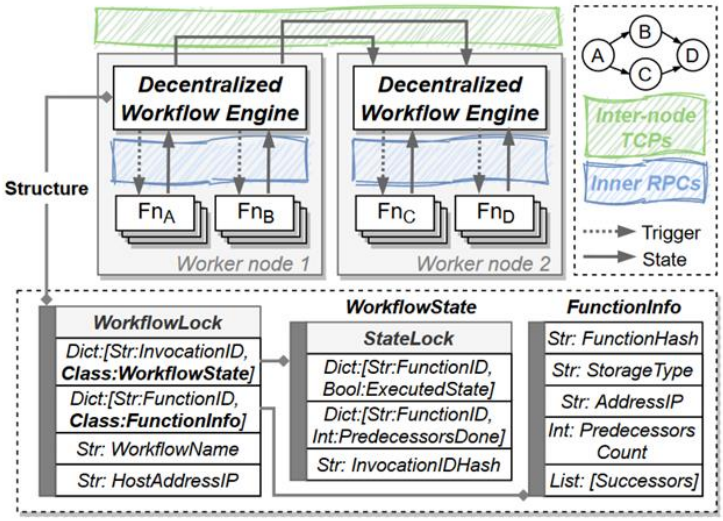

图 2-1 WorkerSP 的函数触发

SAND[6] 则引入了分层消息总线机制，在它的两级层次结构中，有一个分布在主机上的全局消息总线和每个主机上的本地消息总线，若函数实例在本地存在则可通过本地消息总线通知，以便快速触发在同一主机上运行的函数功能。

WUKONG[14]基于 AWS Lambda 实现了一个弱中心化的 DAG 引擎，它结合静态和动态调度，将工作流程划分为子图，并由 Lambda 执行器负责调度和执行分配的任务。然而，WUKONG 在 Lambda 执行器中进行多个函数托管可能会引入额外的安全漏洞。

2.3 数据驱动的函数编排与优化

值得注意的是，上述研究在考虑弱化中心化架构的同时，也在探讨如何通过利用数据本地性来减少数据传输的时延。而另外一些研究则更侧重于数据，他们倾向于考虑以数据为中心的调度策略。 当前的无服务器平台通过遵循高级调用依赖关系来协调和触发函数，但忽略了函数之间的底层数据交换。这种设计在编排复杂的工作流程时既不高效也不易于使用——开发人员往往需要自己管理复杂的功能交互，实现定制化，性能不尽如人意。

SONIC[8]是一种应用感知的数据传递管理器，专为链式无服务器应用设计。它针对无服务器应用中函数之间交换中间数据的问题，提出了一种优化应用性能和成本的解决方案。SONIC 提出了三种数据传递方法：VM-Storage、Direct-Passing 和 Remote-Storage，Storage 通过在同一 VM 上调度发送和接收函数来利用数据

局部性，而 Direct-Passing 则在不同 VM 的函数之间直接复制中间数据，实现数据交换。Remote-Storage 涉及通过远程存储系统上传和下载数据。SONIC 认为没有单一的数据传递方法在所有情况下都是最优的，应该动态地为无服务器工作流的每个边选择最优的数据传递方法，实现数据交换。然而，SONIC 的实现可能需要对无服务器平台进行更深入的修改，以实现其数据传递优化策略，这可能会对平台的通用性和易用性产生影响。

Pheromone[15]在无服务器计算中提出了一种全新的数据中心化的函数编排方法。它通过引入数据桶抽象和数据触发原语，使得开发者可以精细控制函数间的数据交换和触发关系。在执行过程中，函数将生成的中间数据发送到指定的数据桶，数据桶根据预设的触发条件自动触发目标函数并将数据传递给它们。这种设计使得复杂的函数交互变得更加简单和高效。虽然 Pheromone 在处理大型任务时已显示出巨大的可扩展性和性能，但由于 Pheromone 的性能优化仅适用于某些语言运行时，它仅能充分利用直接使用字节数组的语言（比如 Python ctype），以实现零复制数据共享。

此外一些研究[25][6]尝试在同一容器中执行 FaaS 任务的所有功能来减少延迟并提高功能链接的资源效率，而 SpecFaaS[26] 走得更远，它同时考虑数据流和控制流，并认为应用程序中执行的功能序列具有高度可预测性，这种策略基于对 FaaS 应用程序特征的深入分析，允许应用程序中的函数在其控制和数据依赖被充分满足之前，预测性地提前执行。若预测发生错误，FaaS 控制器将取消错误预测的函数及其所有后续函数，同时将数据缓冲区中相应的数据置为无效。有了这种机制，下游函数的执行可以与上游函数的执行并行，从而显著减少应用程序的端到端执行时间。这种同时考虑数据流和控制流的方法为本工作的研究提供了启示。在事件驱动的 FaaS 编排执行系统设计中，本工作将特别关注数据流和控制流的整合。

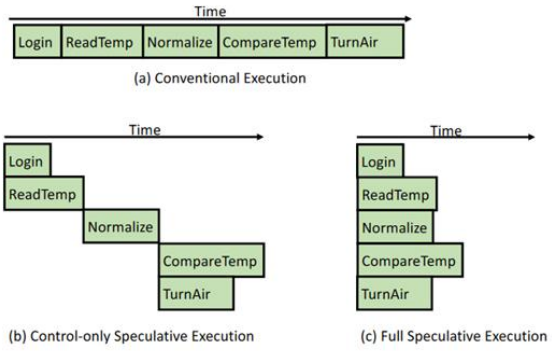

图 2-2 常规执行与推测执行

2.4 事件驱动系统在任务编排中的应用

然而，这些研究并没有从函数的通信模式方面进行深入探究。函数间的通信通常基于请求-响应模型，且依赖于中心化的编排器进行服务间的协调和调度。而采用事件驱动的通信模式，可以大幅度减轻中央调度器的负载，降低任务调度的时延，并且并能够支持复杂的结构。

实际上，许多事件驱动的概念，如触发器、事件条件动作（ECA）甚至复合事件检测，事件驱动机制和触发器在过去被广泛用于构建传统任务编排系统[16, 17, 18, 19]，包括触发器和规则的 ECA 模型非常适合定义表示工作流的有限状态机的转换。Dai 等人[20]提出使用同步聚合触发器来协调大规模并行的数据处理任务，Li 等人[19]利用基于内容发布/订阅系统的复合订阅来提供分布式的基于事件的工作流管理。他们的 PADRES 系统通过基于内容的订阅在复合订阅语言中支持并行化，交替，序列和重复组成。

Soffer 等人[21] 调查了复杂事件处理（CEP）和商业流程管理社区的交叉点，他们清楚地介绍了结合两种模型的现有挑战，并描述了这个领域的最新工作。他们指出“通过 CEP 规则执行商业流程”是一个重大挑战。更相关的研究工作旨在为无服务器函数提供反应式编排，Baldini 等人[22] 提倡为函数编排提供反应式运行时支持，并在 Apache OpenWhisk 之上提供了一个用于顺序组合的解决方案。

事件驱动的无服务器任务编排方式同样面临着一定的挑战，Burckhardt 等人[23]研究对比了持久函数（即代码形式的工作流）和触发器在任务编排中的应用，揭示了使用触发器进行工作流程编排的若干局限性。这些局限性主要体现在：(i) 需要为每一步骤创建不同的队列/目录；(ii) 触发器无法等待多个前置步骤的完成；

(iii) 触发器在错误处理上的适用性不强。Triggerflow [36]提出了一种创新的“富触发器”（Rich Trigger）框架以解决这些问题：通过指定规则来过滤事件，避免创建多个队列；通过聚合事件，以执行多重连接；通过事件重放和检查点，保证容错。并实现了一个基于触发器的支持异构工作流程和高容量工作负载的 FaaS 任务编排框架，支持多种事件源，动作类型，支持第三方过滤器或条件，提供了一个共享的持久性上下文存储库，提供耐用性和容错性。

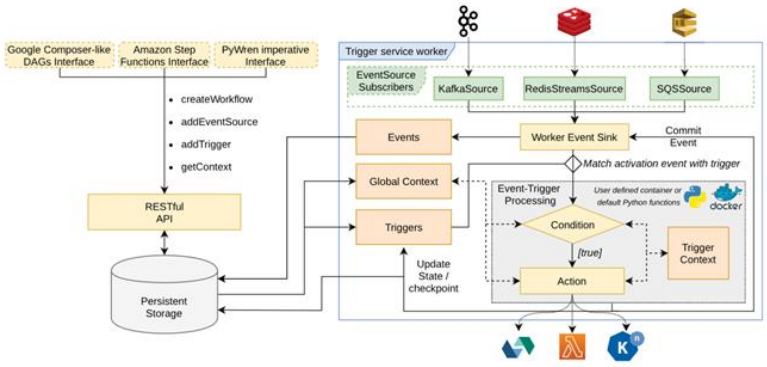

图 2-3 Triggerflow 架构图

最近，云原生计算基金会（CNCF）投入巨大精力制定无服务器工作流的标准规范[24]。他们推出了一种以 YAML 文件形式声明工作流的方法，该文件包含了用于消费 CloudEvents 的描述、事件驱动的无服务器函数调用，以及工作流数据管理和控制流逻辑的状态转换。此设计理念在于制定一种能被多种系统理解的抽象定义，以确保工作流的可移植性，进而避免厂商锁定问题。

3. 课题主要研究内容、预期目标

3.1 研究内容

本课题的研究内容主要围绕在无服务器架构下实现事件驱动的 FaaS 编排执行系统，具体的研究内容如下：

(1)基于事件机制的 FaaS 编排模型。本课题将设计一个基于事件机制的 FaaS 编排模型，该模型将控制流和数据流统一起来，并对事件的结构和属性进行定义。为了使得用户可以自定义事件，工作将设计事件定义接口，并提出相应的规则与限制。此外，工作将探索如何将 DAG 模型的元素映射到事件，以实现与传统任务编排模型的兼容。

(2)事件驱动的函数调度执行机制。本课题将研究如何以事件驱动的方式进行函数的运行时管理，以此来实现对控制流和数据流的高效协同处理。具体而言，需要设计状态追踪与事件生成组件，应考虑数据流的细粒度控制问题。事件接收与函数触发组件，应支持过滤事件，聚合事件等高级事件处理功能。

(3) 事件驱动的 FaaS 编排执行系统实现。基于上述的研究，工作将设计并实现一个事件驱动的 FaaS 编排执行系统。我们将详细阐述系统的架构，并对各个模块进行设计。

综上，本课题的研究内容可以分为以下三个部分：1）基于事件机制的 FaaS 编排模型；2）基于事件驱动模型的函数运行时管理；3）事件驱动的 FaaS 编排执行系统的设计与实现。

3.1.1 基于事件机制的 FaaS 编排模型

在无服务器计算中，DAG（有向无环图）模型由于其自然而直观的方式来表达任务间的依赖关系，已经被广泛接受和采用。然而，在面向无服务器的环境下，DAG 这种静态的、预定义的结构在一定程度上限制了任务编排的复杂度和灵活性，因此，工作提出了一种事件驱动的编排模型，将 DAG 模型转化为事件驱动的编排模型，以便更好地适应无服务器环境。

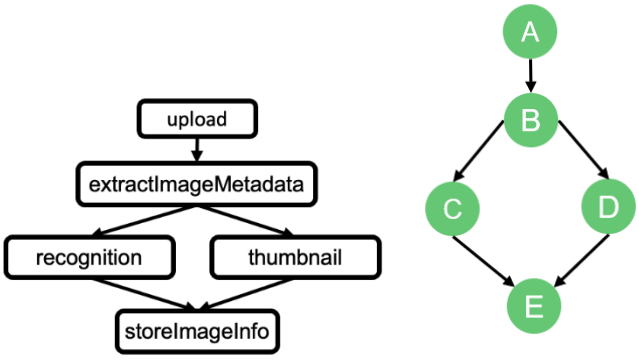

图 3-1 DAG 模型示例——图像识别应用

FaaS 编排模型的设计：本工作将设计和定义一个统一的事件驱动模型，该模型将综合考虑控制流和数据流，定义事件的结构和属性。事件将成为调度和执行的基本单位，从而使得系统可以在事件级别上进行更精细的管理和调度。

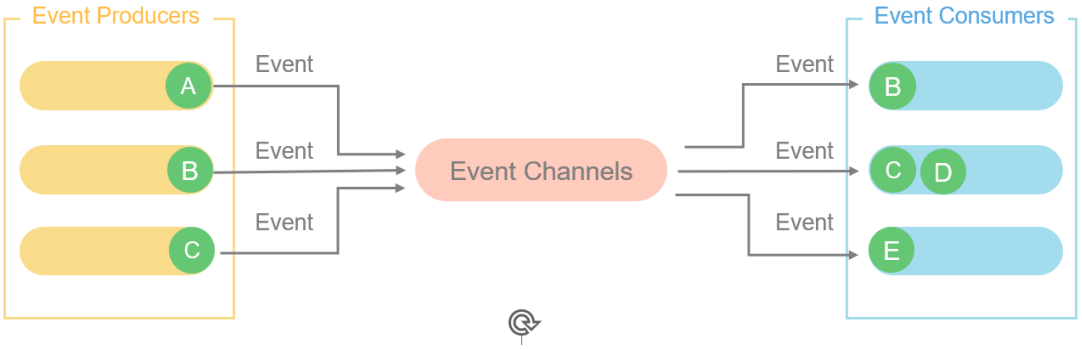

图 3-2 事件驱动系统中函数的触发

DAG 到事件驱动编排模型的转换：为了在实际环境中应用事件驱动模型，需要将现有的 DAG 模型转化为事件驱动模型。工作将研究如何将 DAG 中的节点和边映射到事件，以及如何将数据映射到事件，以实现从 DAG 模型到事件驱动模型的平滑转换。

3.1.2 事件驱动的函数调度执行机制

在构建事件驱动模型的 FaaS 编排执行系统中，管理和调度函数的运行是至关重要的一环。现有的系统通常围绕控制流展开调度。实际上，对数据流的高效管理和调度可以显著提高任务的执行效率，并使得函数的执行能够得到有效的重叠[37]，从而提高整体的执行性能。

然而，以数据流为中心的研究工作又往往对控制流的考虑不足，而且这类系统的架构和主流的 FaaS 平台差异较大，对用户的学习成本较高。因此，工作希

望在不大幅改变用户习惯的前提下，研究函数调度执行机制，以事件的形式，将控制流和数据流统一管理，触发函数的执行。

状态追踪与事件生成组件：关注数据流和控制流状态的实时监控，以生成相应事件。精确的状态追踪是触发事件的关键，需要结合实时数据和控制流信息，创建适应的事件。组件的设计和实现将影响事件驱动模型的精度和效率。

事件接收与函数触发组件：对事件进行处理并根据触发器动作列表（Trigger-Action List,TAL）触发相应动作。这需要高效的查询和处理机制，以在满足触发条件时迅速准确地执行函数。此组件的设计涉及到事件处理能力和系统性能。

这两个组件共同构成了函数调度执行机制，对于构建高效的事件驱动 FaaS 任务编排系统起到关键作用。

3.1.3 事件驱动的 FaaS 编排执行系统的设计与实现

事件驱动的 FaaS 编排执行系统的设计与实现是研究的重要内容，旨在实现一种高效、可扩展且用户友好的事件驱动系统。这需要根据前述的事件驱动模型和函数运行时管理机制，进行系统的设计与实现。

系统架构：首先要设计出符合事件驱动模型和基于事件的函数运行时管理机制的系统架构。考虑系统的模块划分，包括但不限于事件管理模块，调度模块等。同时也将考虑系统的可扩展性，以支持大规模的 FaaS 任务处理需求。

模块设计：在明确了系统的架构之后，将对各个模块进行详细的设计。例如，设计状态追踪与事件生成模块，负责事件流的生成、管理和调度；设计事件接收与函数触发模块，负责基于事件触发函数的执行和管理；

系统实现：最后，工作将根据前述的系统架构和模块设计，进行系统的实现。首先将选择适合的编程语言和框架，编写代码，实现系统的各个功能。在系统实现过程中，将注重代码的可读性和可维护性，以支持未来的功能扩展和维护。

通过上述的工作，本工作将实现一种事件驱动的 FaaS 编排执行系统，它将能够有效地处理和调度大规模的 FaaS 任务，提高系统的执行效率，减少数据传输和计算延迟，同时还具有较好的用户体验。

3.2 预期目标

预期目标主要分为三个方面：

（1）基于事件机制的 FaaS 编排模型。建立一个统一的事件驱动编排模型，以实现对控制流和数据流的统一管理。这个模型将通过定义事件的结构和属性，使得事件可以准确地代表函数的执行和数据的传递。

（2）事件驱动的函数调度执行机制。设计并实现状态追踪与事件生成组件，事件接收与函数触发组件，以实现对事件流的正确调度。同时选择合适的事件路由和分发机制，它能够根据事件的属性和系统的状态，将事件准确地路由到正确的处理函数。

（3）事件驱动的 FaaS 编排执行系统的设计与实现。根据前述的 FaaS 编排模型和函数运行时管理机制，设计并实现一个高效、可扩展且用户友好的事件驱动 FaaS 编排执行系统，从而推动 FaaS 任务在更广泛的应用场景中的应用。

4. 拟采用的研究方法、技术路线、实验方案及可行性分析

4.1 研究方法和技术路线

4.1.1 基于事件机制的 FaaS 编排模型

在这一部分中，工作的核心目标是建立一个基于事件驱动的任务编排模型，尝试将 FaaS 任务描述为“触发器-动作列表”（Trigger-Action List，简称 TAL）TAL 的基本定义如下：

定义 1 事件（event）：事件为一个四元组，其中 e 是事件的具体类型（如 A 函数执行结束，B 函数执行结束等），t 是事件类型（控制事件或数据事件），Δc 是上下文的变化，如果没有变化，这个字段可以为空。P 是一个属性集合，包含了 CNCF CloudEvents version 1.0 定义的事件的必选属性。

$$E = (e, t, \Delta c, P) \tag{4-1}$$

定义 2 触发器（Trigger）：触发器为一个三元组 其中 E 是一个事件，c 是任务初始化时的上下文信息，ϕ 是一个布尔函数，表示触发器的条件。

$$T = (E, c, \phi) \tag{4-2}$$

定义 3 动作（Action）：动作为一个函数 f，表示动作的执行。

$$A = f \tag{4-3}$$

定义 4 任务（Task）：任务为一个列表，列表中的每一项都是一个二元组(T,A)，其中 T 是一个触发器，A 是该触发器激活时应执行的动作。

$$Task = \{(T_1, A_1), (T_2, A_2), \dots, (T_n, A_n)\} \tag{4-4}$$

定义 5 动作执行（Action execution）：在这个模型中，一个任务中的动作 A 的执行条件是：在任务中，只要存在一个触发器 T 使得 ϕ(E,c)=True，那么与该触发器关联的动作 A 就会执行。

$$\exists (T, A) \in Task, \phi(E, c) = True \Rightarrow A() \tag{4-5}$$

这种模型为处理复杂的执行场景提供了更大的灵活性，同时保持了原有的执行逻辑。

为将现有的有向无环图（DAG）模型转换为 TAL 模型，需要实现一个转换器，按节点映射到动作、边映射到触发器的原则进行转换。此外，在以 DAG 方

式描述的工作流中，数据的准备通常是不可见的，为了在函数级别实现对数据准备完成事件的声明，需要设计并实现一个专门的编程接口。此接口需要提供一种机制，使得函数能够在数据准备完成时发出一个明确的信号。这种设计将数据准备的过程显式地表示，而不是像传统的 DAG 模型那样将其隐式地包含在函数的执行过程中。

这个设计的主要目标是实现控制流与数据流的解耦。在传统的 DAG 模型中，控制流（即函数执行的顺序）通常与数据流（即数据的传递和处理）紧密耦合。在新模型试图将这两者分开，使得函数可以在其依赖的数据准备好后立即执行，而不必等待其他不相关的函数。

4.1.2 事件驱动的函数调度执行机制

工作计划在每个节点部署事件处理模块，以优化事件流管理。该模块主要由两个组件构成：状态追踪与事件生成组件（EventGen）和事件接收与函数触发组件（EventSink）。

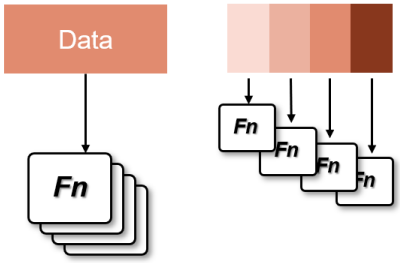

图 4-1 动态触发事件的执行优化

EventGen 组件包括 API 接口、数据状态追踪器、事件产生规则集和事件发布者。API 接口支持用户定义数据的读写操作和触发条件。数据状态追踪器负责跟踪数据状态，并在数据达到预设条件时，基于规则集，通过事件发布者发布对应的事件。EventGen 主要支持三种类型的事件触发：数据就绪触发、动态触发和条件触发。其中，动态触发针对并行循环（foreach）结构，能够动态生成事件，实现边切分数据、边调用循环函数；条件触发则依据设定的条件（如数据数量达阈值或预设时间间隔）生成相应事件。

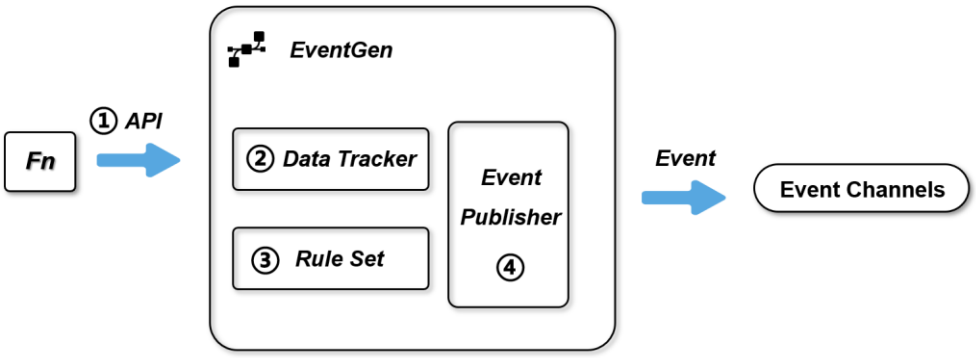

图 4-2　EventGen 组件示意图

EventSink 组件由函数触发器、本地触发器动作列表和事件订阅者组成。事件订阅者接收事件通道的事件并交给函数触发器，函数触发器会根据本地触发器动作列表中的触发关系找到并执行对应的动作，该过程通常通过 Restful API 方式实现。

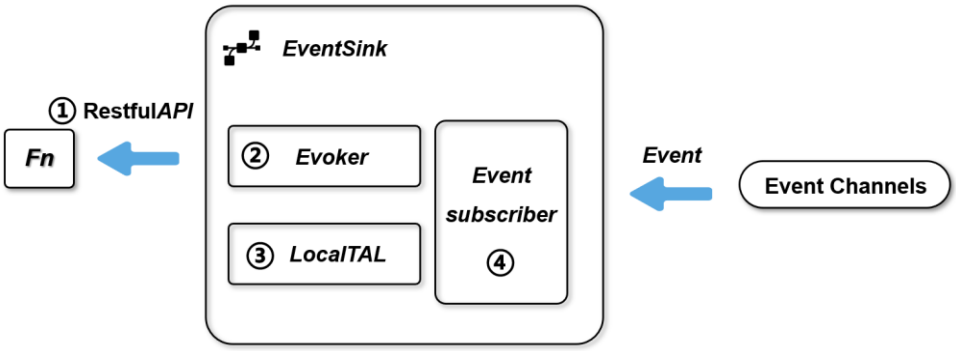

图 4-3　EventSink 组件示意图

在确定了事件的管理机制后，如何对这些事件进行有效的路由和分发值得重点研究。工作考虑采用发布/订阅（pub/sub）机制，并利用消息队列来实现这一部分的功能。

4.1.3　事件驱动的 FaaS 编排执行系统的设计与实现

事件驱动的 FaaS 编排执行系统总体分为控制节点和执行节点两部分。控制节点包括任务调度器、任务管理器、资源管理器和事件处理器，主要负责管理和维护 FaaS 任务，以及资源的监控和调度；执行节点包括任务执行器、数据代理、事件处理器和资源报告器，主要负责任务的执行和资源的监控。控制节点的模块

及其详细功能如下：

（1）任务调度器（Scheduler）：负责将 FaaS 任务基于细粒度的数据依赖进行划分放置，它将任务管理器输出的任务模型和资源情况作为输入，对函数进行划分，并选择合适节点进行部署。

（2）任务管理器（Manager）：用户为 FaaS 任务开发者，任务管理器帮助用户维护任务的定义、状态和生命周期，并将定义信息转换为触发器-动作列表（TAL）的形式进行存储。

（3）资源管理器（Resource Monitor）：主要监控执行节点的资源使用情况，包括 CPU、内存和存储。在资源过载的情况下，资源管理器可以再次调度任务。

（4）事件处理器（Event Handler）：负责接收并处理来自消息队列的事件，并将新的事件发布到消息队列。事件处理器维护控制节点的事件驱动机制。

计算节点的模块及其详细功能如下：

（1）任务执行器（Task Executor）：位于执行节点，执行具体的任务，并在任务执行完成或失败后生成相应的控制相关事件，然后直接将这些事件发布到消息队列中。

（2）状态追踪与事件生成组件（EventGen）：负责根据函数执行状态与数据状态生成相关事件，基于规则集，通过事件发布者发布对应的事件。

（3）事件接收与函数触发组件（EventSink）：接收来自消息队列的事件，并将事件，交给函数触发器，函数触发器会根据本地 TAL 中的触发关系找到并执行对应的动作。

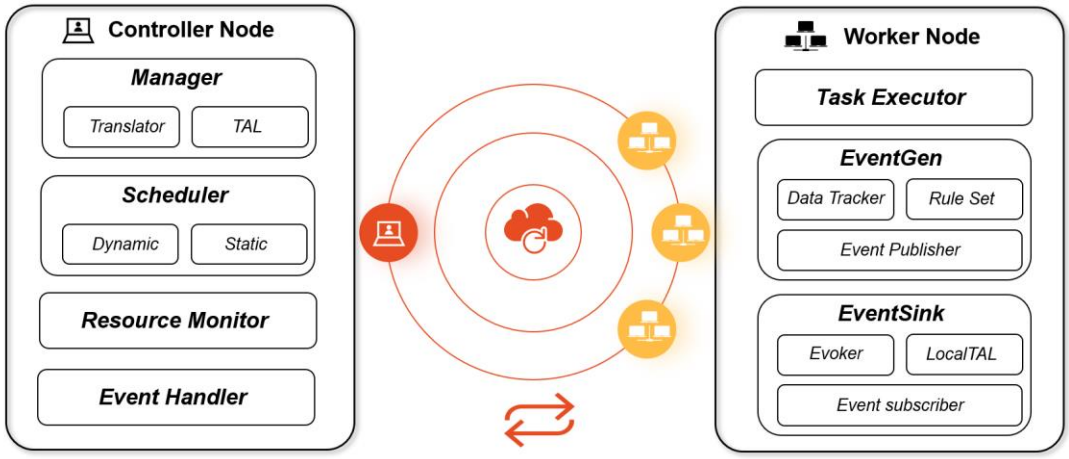

图 4-4　事件驱动的 FaaS 编排执行系统架构图

4.2 实验方案

大部分现有的平台和研究都将 FaaS 任务编排为工作流的形式。实验首先需要对工作流数据集进行收集，这将通过选取并整理来自文献（如 FaaSFlow）或公开网络资源（如 Kaggle）的数据完成。

实验将从三个方面评估系统性能：系统可用性、负载处理能力以及复杂任务处理能力。

系统可用性的测试将使用前述收集的大规模工作流数据集，以检验系统能否正确划分和调度各个工作流应用；负载处理性能测试将观察系统在不同负载条件下的调度延迟、并行处理能力和吞吐量等指标；系统处理复杂任务能力测试将运行选定的复杂任务（如涉及到“shuffle”操作的大数据处理任务），评估系统处理这些任务的时延、并行度和总体执行时间。

此外，比较评估也是关键步骤，工作将从商业或开源平台以及相关研究中选取与本系统功能相似的已有系统（如 Google Workflows、Triggerflow 等）作为基准系统，在相同的条件下，分别运行基准系统和本系统，记录并对比性能指标。

最后需要收集所有实验结果，进行数据分析和结果解释，并根据实验结果，提出系统优化的建议和策略。

4.3 可行性分析

首先，本论文在前期已进行了较为充分的文献资料收集和调研工作，对相关工作的优缺点有较为充分的认识。同时，本论文工作在相关工作的基础上，积累一定的系统设计与研究经验，为论文提出的方法及设计打下了基础。

其次，实验室提供了完备的设施和资源，包括硬件设备、软件平台以及网络资源等，能够满足进行相关研究和实验的需要。

最后，本论文有实验室富有相关科研经验的科研人员指导，为论文的顺利进行提供了大量的理论和技术支持。

5. 已有科研基础与所需的科研条件

5.1 已有基础

（1）目前已经调研了大量与、FaaS 系统设计、任务编排、事件驱动模型、数据流调度相关的科研文献，积累了一定的理论知识和研究基础。

（2）有大量成熟的开源工具可以为系统设计提供支持，这在一定程度上节省了开发时间，避免不必要的重复工作。

（3）实验室可提供边缘节点计算存储等服务资源，如 GPU、Raspberry Pi 等并附有相关使用文档和技术支持。

（4）实验室已经有一些完成的研究工作与本工作的研究方向相关，为研究提供了宝贵的参考资料和启发。

（5）实验室配备有完备先进的科研环境，同时有经验丰富的优秀的导师们的指导，和优秀的组内同学的相互协助。

5.2 所需科研条件

通过实验室提供的电子资源检索权限，查阅相关文献；分布式环境的设备支持；项目运行所需要的服务器资源等。

6. 研究工作计划与进度安排

表 6.1 研究工作计划与进度安排表

开始时间	结束时间	任务安排
2023 年 7 月	2023 年 8 月	基于事件机制的 FaaS 编排模型设计
2023 年 8 月	2023 年 10 月	事件驱动的函数调度执行机制实现
2023 年 10 月	2024 年 1 月	事件驱动的 FaaS 编排执行系统实现
2024 年 1 月	2024 年 2 月	数据收集与系统测试
2024 年 2 月	2024 年 4 月	总结工作，形成论文

参考文献

[1] Baldini I, Castro P, Chang K, et al. Serverless computing: Current trends and open problems[J]. Research advances in cloud computing, 2017: 1-20.

[2] Unknown Author. (N.D.). Apache OpenWhisk Composer. Available at: https://github.com/apache/openwhisk-composer.

[3] Unknown Author. (N.D.). AWS Step Functions. Available at: https://aws.amazon.com/step-functions/.

[4] Unknown Author. (N.D.). Google Cloud Composer. Available at: https://cloud.google.com/composer.

[5] KNIX Serverless. https://github.com/knixmicrofunctions/knix/.

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

[7] Kotni S, Nayak A, Ganapathy V, et al. Faastlane: Accelerating function-as-a-service workflows. In Proc. USENIX ATC, 2021.

[8] Mahgoub A, Wang L, Shankar K, et al. SONIC: Application-aware Data Passing for Chained Serverless Applications[C]//2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021: 285-301.

[9] Müller I, Marroquín R, Alonso G. Lambada: Interactive data analytics on cold data using serverless cloud infrastructure. In Proc. ACM SIGMOD, 2020.

[10] Unknown Author. (2021). Durable Functions is an extension of Azure Functions that lets you write stateful functions in a serverless compute environment. Available at : https://www.alibabacloud.com/product/serverless-workflow.

[11] Unknown Author. (2021). AlibabaServerlessWorkflow: Visualization, free orchestration, and Coordination of Stateful Application Scenarios. Available at : https://www.alibabacloud.com/product/serverless-workflow.

[12] Unknown Author. (2021). Fission Workflows: Fast, reliable and lightweight function composition for serverless functions. Available at: https://github.com/fission/fission-workflows.

[13] 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.

[14] Carver B, Zhang J, Wang A, et al. Wukong: A scalable and locality-enhanced framework for serverless parallel computing[C]//Proceedings of the 11th ACM Symposium on Cloud Computing. 2020: 1-15.

[15] 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 (NSDI 23). 2023: 1489-1504.

[16] A. Geppert, D. Tombros, Event-based distributed workflow execution with EVE, in: Middleware'98, Springer, 1998, pp. 427–442.

[17] Chen W, Wei J, Wu G, Qiao X. Developing a concurrent service orchestration engine based on event-driven architecture[C]//OTM Confederated International Conferences "On the Move to Meaningful Internet Systems". Springer, 2008: 675-690.

[18] Binder W, Constantinescu I, Faltings B. Decentralized orchestration of composite web services[C]//2006 IEEE International Conference on Web Services (ICWS'06), IEEE, 2006: 869–876.

[19] Li G, Jacobsen H-A. Composite subscriptions in content-based publish/subscribe systems[C]//ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing. Springer, 2005: 249-269.

[20] Dai D, Chen Y, Kimpe D, Ross R. Trigger-based incremental data processing with unified sync and async model[J]. IEEE Trans. Cloud Comput., 2018.

[21] P. Soffer, A. Hinze, A. Koschmider, H. Ziekow, C. Di Ciccio, B. Koldehofe, O. Kopp, A. Jacobsen, J. Sürmeli, W. Song, From event streams to process models and back: Challenges and opportunities, Inf. Syst. 81 (2019) 181–200.

[22] I. Baldini, P. Cheng, S.J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, O. Tardieu, The serverless trilemma: Function composition for serverless computing, in: Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Onward! 2017, 2017, pp. 89–103

[23] S. Burckhardt, C. Gillum, D. Justo, K. Kallas, C. McMahon, C.S. Meiklejohn, Serverless workflows with durable functions and netherite, 2021, arXiv: 2103.00033.

[24] Cloud Native Computing Foundation. (N.D.). Serverless Workflow. Available at: https://serverlessworkflow.io/.

[25] S. Kotni, A. Nayak, V. Ganapathy, and A. Basu, "Faastlane: Accelerating Function-as-a-Service Workflows," in Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC '21), 2021.

[26] Stojkovic J, Xu T, Franke H, et al. SpecFaaS: Accelerating Serverless Applications with Speculative Function Execution[C]//2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023: 814-827.

[27] Guo J, Chang Z, Wang S, Ding H, Feng Y, Mao L, Bao Y. Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces[C]//ACM IWQoS '19. 2019.

[28] Unknown Author. (N.D.). Developing Convex Optimization Algorithms in Dask parallel: math is fun. Available at: https://matthewrocklin.com/blog/work/2017/03/22/daskglm-1.

[29] Unknown Author. (N.D.). Developing Convex Optimization Algorithms in Dask parallel: math is

fun. Available at: https://matthewrocklin.com/blog/work/2017/03/22/daskglm-1.

[30] Cortes C, Vapnik V. Support-Vector Networks[J]. Mach. Learn., 20, 3 (Sept. 1995): 273–297. DOI: https://doi.org/10.1023/A:1022627411411

[31] Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, Hang Cui, and Edward Y. Chang. 2008. Parallelizing Support Vector Machines on Distributed Computers. In Advances in Neural Information Processing Systems 20, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis (Eds.). Curran Associates, Inc., 257264. http://papers.nips.cc/paper/3202-parallelizing-support-vectormachines-on-distributed-computers.pdf

[32] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[33] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12). 2012: 15-28.

[34] Ousterhout K, Wendell P, Zaharia M, et al. Sparrow: distributed, low latency scheduling[C]//Proceedings of the twenty-fourth ACM symposium on operating systems principles. 2013: 69-84.

[35] Crist J. Dask & Numba: Simple libraries for optimizing scientific python code[C]//2016 IEEE International Conference on Big Data (Big Data). IEEE, 2016: 2342-2343.

[36] López P G, Arjona A, Sampé J, et al. Triggerflow: trigger-based orchestration of serverless workflows[C]//Proceedings of the 14th ACM international conference on distributed and event-based systems. 2020: 3-14.

[37] Krintz C, Calder B, Lee H B, et al. Overlapping execution with transfer using non-strict execution for mobile programs[J]. ACM SIGPLAN Notices, 1998, 33(11): 159-169.