Computer Science and Application 计算机科学与应用, 2018, 8(12), 1936-1943
Published Online December 2018 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2018.812215

Hans汉斯

Research and Implementation of Distributed Task Concurrent Coordinated Based on ODL

Guojun Chen[1], Ningsheng Fang[1,2]

[1]Jiangsu Key Construction Laboratory of IOT Application Technology, Wuxi Taihu University, Wuxi Jiangsu
[2]Southeast China University, Nanjing Jiangsu
Email: cgj3721@163.com

Received: Dec. 10th, 2018; accepted: Dec. 21st, 2018; published: Dec. 28th, 2018

Abstract

With the popularity of the Internet and more complicated network system, network operation requires the integration and collaboration with other IT operations, which causes the difficulties in network deployment: troublesome updating network, too much manual operation, overwhelmed network administrator. Someone tried to change the structure of a network or to launch a new network device for the above problems, but this improvement has failed to solve the problem of the existing network, them the software defined network (SDN) arises at the historic moment. In the operation process of SDN, it often applies distribution to improve the efficiency. Under distributed environment there must be distributed multi-task concurrent, and multi-task concurrent will lead to difference in time and data for its concurrency features, namely distributed concurrent task coordination. In this paper, the purpose of this research is to use SDN transparent open source framework OpenDaylight to study the concurrency and coordination of distributed tasks.

Keywords

Software Defined Network, OpenDayLight, Distributed, Task Concurrent, Coordinated

基于ODL的分布式任务并发协调一致的研究

陈国俊[1]，方宁生[1,2]

[1]无锡太湖学院，江苏省物联网应用技术重点建设实验室，江苏 无锡
[2]东南大学，江苏 南京
Email: cgj3721@163.com

收稿日期：2018年12月10日；录用日期：2018年12月21日；发布日期：2018年12月28日

文章引用: 陈国俊, 方宁生. 基于 ODL 的分布式任务并发协调一致的研究[J]. 计算机科学与应用, 2018, 8(12): 1936-1943.
DOI: 10.12677/csa.2018.812215

摘　要

随着互联网的普及，网络体系越来越复杂，网络操作需要与其它IT操作的集成与协作，导致网络部署困难更新麻烦、手动操作过多，网络管理员分身乏术[1]。对于以上问题，优化网络结构、网络设备更新是网络研究者们探索的课题，进而出现了软件定义网络(简称SDN)的概念和方法。在SDN运行过程中为提高其效率经常会应用到分布式，分布式环境下必然存在着分布式的多任务并发，而多任务的并发因其并发特性将会导致时间和数据上的不一致，即分布式任务并发协调不一致。本论文即以此为目的展开研究，采用SDN透明化的开源框架OpenDayLight研究分布式任务并发协调一致的问题。

关键词

软件定义网络，OpenDayLight，分布式，任务并发，协调一致

1. 引言

分布式环境下多任务的并发场景经常出现，分布式任务并发不一致即在分布式环境下因多任务并发造成任务到达时间以及数据的不一致，而时间及数据的不一致将导致分布式任务的失败或集群效率及性能不高，因此必须解决分布式任务并发协调不一致的问题[2]。

在 ODL 集群中同样存在着分布式任务并发协调不一致的问题，包括在时间上的不一致以及在数据上的不一致，而 ODL 框架中目前没有对应的解决方案。

为解决 ODL 集群中分布式任务并发在时间上协调不一致的问题可以运用 ZooKeeper，ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，本论文将 ZooKeeper 部署到 ODL 集群中，通过 ZooKeeper 编码实现屏障使得分布式任务并发在时间上实现协调一致。

对于 ODL 集群中分布式任务并发在数据上协调一致的问题，可以通过本论文中提出的数据提交算法解决。

2. 相关背景介绍

本论文主要工作是基于 OpenDayLight 建立集群，在此基础上利用 ZooKeeper 及研究提出的数据提交算法解决分布式任务并发协调不一致的问题,下面对 OpenDayLight 及分布式任务并发进行简要介绍。

2.1. OpenDayLight

OpenDayLight 架构主要由应用服务层、控制平面层、南向接口层和数据平面层四层构成[3]。

OpenDayLight 为应用(App)提供开放的北向 API。支持 OSGi 框架和双向的 REST 接口。OSGi 框架提供给与控制器运行在同一地址空间的应用，而 REST API 则提供给运行在不同地址空间的应用。所有的逻辑和算法都运行在应用中[3]。

控制平面主要包含了基本网络服务和一些附加的网络服务，这些附加服务都可以通过插件的形式安装加载，这增加了 OpenDayLight 的灵活性，当然了其稳定性也是显而易见的[3]。

ODL 控制器采用 OSGi 框架，OSGi 框架是面向 Java 的动态模型系统，它实现了一个优雅、完整和动态的组件模型，应用程序(Bundle)无需重新引导可以被远程安装、启动、升级和卸载，通过 OSGi 捆绑可以灵活地加载代码与功能，实现功能隔离，解决了功能模块可扩展问题，同时方便功能模块的加载与协同工作。自 Helium 版本开始使用 Karaf 架构，作为轻量级的 OSGi 架构，相较于早前版本的 OSGi 提升了交互体验和效率，当然其特性远不仅仅于此。

ODL 控制平台引入了 SAL (服务抽象层)，SAL 北向连接功能模块，以插件的形式为之提供底层设备服务，南向连接多种协议，屏蔽不同协议的差异性，为上层功能模块提供一致性服务，使得上层模块与下层模块之间的调用相互隔离。SAL 可自动适配底层不同设备，使开发者专注于业务应用的开发[4]。

2.2. 分布式任务并发协调一致

分布式系统中的每个节点既独立工作，又与所有其他节点并行工作。每个节点多于一个进程(执行程序)，每个进程多于一个线程(并行执行任务)，可在系统中充当组件。大多数组件具有反应性，对来自用户的命令和来自其他组件的消息不断地进行响应。像操作系统一样，分布式系统旨在避免终止，因此应始终保持至少部分可用的状态。分布式并发控制作为分布式事务管理的基本任务之一，其目的是保证分布式数据库系统中多个事务高效而正确地并发执行。

目前对于分布式并发控制的方法为：基于锁(Locking)机制的并发控制方法、基于时间戳的分布式并发控制、并发控制的乐观法以及统一触发途径等方法。

3. 分布式任务并发协调一致的研究与实现

在 ODL 框架中，分布式任务的并发场景下，在时间上因网络或网速等原因分布式任务到达 leader 节点的时间会存在差异，这样会导致分布式任务时间上不一致的问题，同时，分布式任务并发在数据提交时由于数据提交的失败或回滚等问题也会造成分布式任务并发在数据上的不一致，而分布式任务并发在时间和数据上的不一致会给 ODL 集群带来性能及效率上的重大损失。解决分布式任务并发协调一致问题是相当有必要的。

3.1. 分布式任务并发在时间上协调一致

分布式任务在并发时，因为网络、网速等原因很有可能会产生任务在到达时间上不一致的情况，这样的场景会导致任务无法良好的协同进行，导致任务的失败或错误。针对分布式任务并发在时间上协调不一致的问题，本论文依赖一种可靠的、可扩展的、分布式的、可配置的、开放源码的分布式应用程序协调服务——ZooKeeper，通过 ZooKeeper 实现屏障使得分布式任务并发在时间上协调一致[5]，使分布式任务的并发受到屏障的控制，保证分布式任务时间上的一致性，进而解决分布式任务并发在时间上协调不一致的问题。对应的原理图见图 1 所示：

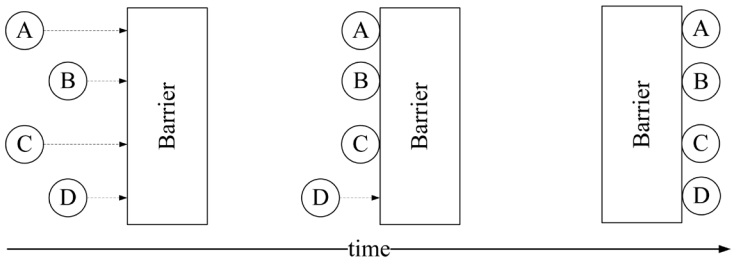

Figure 1. ZooKeeper implementing the barrier schematic
图 1. ZooKeeper 实现屏障原理图

采取的方案就是用一个 Node 作为 Barrier 的实体，需要被 Barrier 的任务通过调用 exists()检测这个 Node 的存在，当需要打开 Barrier 的时候，删掉这个 Node，ZooKeeper 的 watch 机制会通知到各个任务可以开始执行[5]。伪代码如下：

1) 客户端在屏障节点上调用 ZooKeeper API exists()，watch 设置为 true.

2) 如果 exists()返回 false，屏障消失，客户端可以推进的自己的工作。

3) 否则 exists()返回 true，客户端等待屏障节点上监听事件的到来。

4) 如果监听事件被触发，客户端重新执 exists()，再一次重复上述 1~3 步，直到屏障节点被移除[6]。

对应的流程图见图 2 所示：

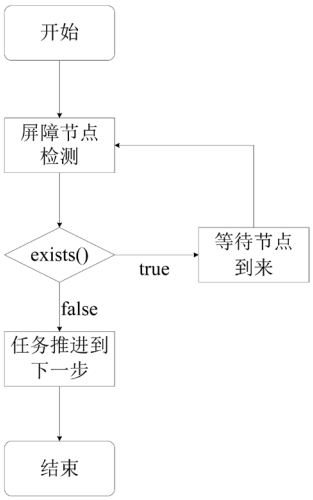

Figure 2. ZooKeeper implementing the barrier flow chart
图 2. ZooKeeper 实现屏障流程图

基于以上分析，可以发现通过 ZooKeeper 实现屏障，让屏障成为分布式任务推进的控制点，只有当所有任务都到达时屏障才消失，任务推进到下一步，从而解决分布式任务并发在时间上协调不一致的问题。

3.2. 分布式任务并发在数据上协调一致

分布式任务，因其分布式的特点，每次数据的提交失败或数据回滚等情况的出现导致其他节点的数据未做到同步，每个节点虽然可以知晓自己的操作时成功或者失败，却无法知道其他节点的操作的成功或失败，因而导致数据不一致的情况。

对于分布式任务并发在数据上协调不一致的问题，通过论文中提出的数据提交算法来控制和确保数据提交在失败或回滚等情况下也能达到分布式任务在数据上的协调一致[7]，从而解决分布式任务在数据上协调不一致的问题。

当一个事务跨越多个节点时，为了保持事务的 ACID 特性，需要引入一个作为中间者的角色来统一掌控所有节点(称作参与者)的操作结果并最终指示这些节点是否要把操作结果进行真正的提交。数据提交算法思路可以概括为：参与者将操作成败通知中间者，再由中间者根据所有参与者的反馈情报决定各参与者是否要提交操作还是中止操作。数据提交算法可以分为两个阶段，第一步为投票阶段，主要获得参与者就绪情况，第二步为事务提交阶段，该阶段是根据数据提交请求情况及参与者就绪情况进行数据的提交或回滚。

具体步骤如下：

投票阶段：

1) 中间者节点向所有参与者节点询问是否可以执行提交操作(vote)，并开始等待各参与者节点的响应。

2) 参与者节点执行询问发起为止的所有事务操作，并将 Undo 信息和 Redo 信息写入日志。

3) 各参与者节点响应协调者节点发起的询问。如果参与者节点的事务操作实际执行成功，则它返回一个“YES”消息；如果参与者节点的事务操作实际执行失败，则它返回一个“NO”消息。根据各参与者的反馈情况来决定最终是否可以进行事物提交操作[7]。

数据提交算法投票阶段流程图见图 3 所示：

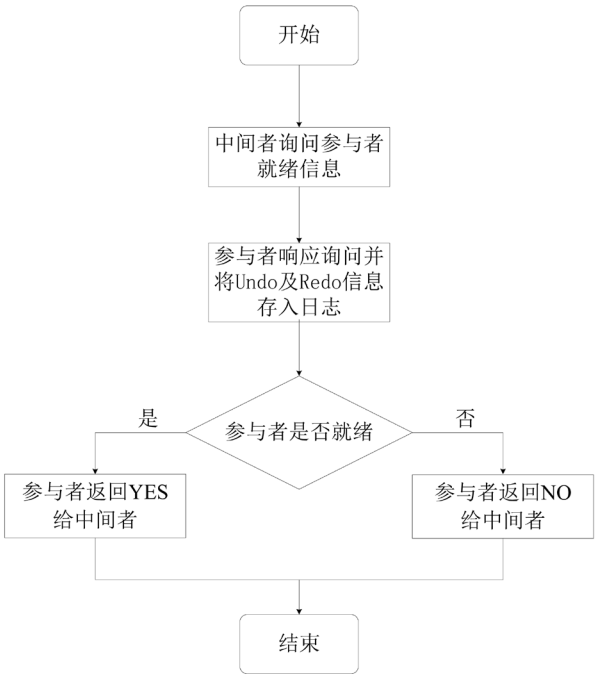

Figure 3. Data submission algorithm voting stage flow chart
图 3. 数据提交算法投票阶段流程图

事务提交阶段：

1) 将中间者节点备份，防止中间者发生故障导致阻塞。当中间者节点从所有参与者节点获得的相应消息都为“YES”时，协调者节点向所有参与者节点发出“正式提交(COMMIT)”的请求。

2) 参与者节点接受到 COMMIT 请求后向中间者节点发送“ACK”消息。

3) 统计参与者返回“ACK”的个数，若返回的个数等于参与者的个数，则所有参与者节点正式完成数据提交操作，并释放在整个事务期间内占用的资源。

4) 如果任一参与者节点在第一阶段返回的响应消息为“NO”，或者中间者节点在第一阶段的询问超时之前无法获取所有参与者节点的响应消息时，中间者节点向所有参与者节点发出“回滚操作(rollback)”的请求。

5) 参与者节点利用之前写入的 Undo 信息执行回滚，并释放在整个事务期间内占用的资源并向协调者节点发送“ACK”消息[8]。

数据提交算法事务提交阶段流程图见图 4 所示：

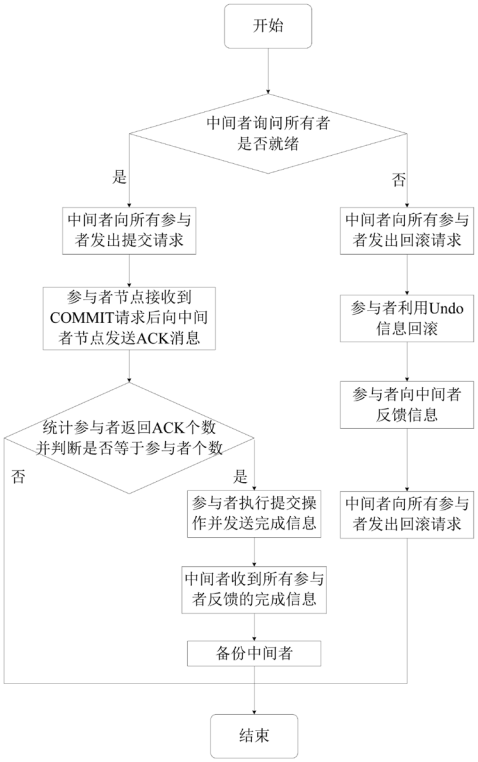

Figure 4. Data submission algorithm transaction submission phase flow chart
图 4. 数据提交算法事务提交阶段流程图

中间者作为数据提交及回滚的控制者，使数据无论在提交请求失败、参与者未就绪或中间者失效等情况下都可以保证数据在各节点的一致性，使数据在分布式任务并发的场景下可以保证数据的一致性，从而解决 ODL 集群中分布式任务并发在数据上不一致的问题。

4. 分布式任务并发协调一致测试结果及分析

本节将根据上文所提出的用 ZooKeeper 实现屏障解决分布式任务并发在时间上不一致的问题及数据提交算法来确保分布式任务并发在数据上的一致性，对其进行测试及分析。

通过实验数据绘制如下折线图，可以发现，当无屏障时，任务到达的时间相差较大，对比下，在有屏障时，任务到达的时间波动很小，证明用 ZooKeeper 实现屏障确实可以达到分布式任务并发在时间上协调一致的预期。分布式任务并发一致在时间上的对比见图 5 所示：

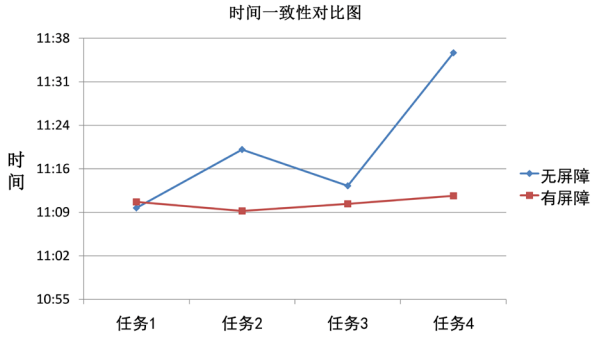

Figure 5. Distributed task concurrent consistent time contrast line graph
图 5. 分布式任务并发一致时间对比折线图

在没有运用数据提交算法时从 leader 节点向四个 follower 节点分别发送相同的数据，同时将第四个 follower 的虚拟机关掉(模拟参与者未就绪的场景)，之后从各个 follower 节点查看数据执行时间。对比结果见图 6 所示：

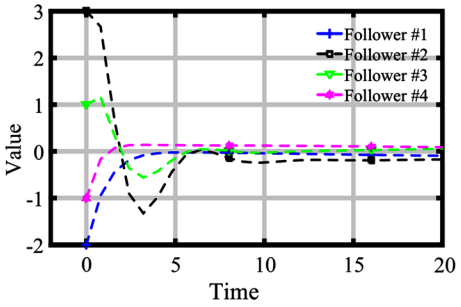

Figure 6. Distributed task data concurrency inconsistent execution time comparison diagram
图 6. 分布式任务数据并发不一致执行时间对比图

通过对比图 6 可以看出 follower 节点第二、第三个的数据执行时间和其他 follower 节点的数据执行时间是不一致的，即分布式任务在并发时数据会因为执行时间不同而出现不一致的场景。

在运用数据提交算法后，因为数据提交前会询问各参与者是否就绪，若未就绪则数据回滚且不会执行后续操作。并且在数据执行提交命令前会统计参与者返回“ACK”的个数，若“ACK”个数和参与者不等，说明存在参与者未收到提交请求的可能性，此时不执行提交操作，否则执行。且为中间者进行备份，防止中间者故障导致阻塞，从而提高效率。对比上述 follower 节点数据执行时间见图 7 所示：

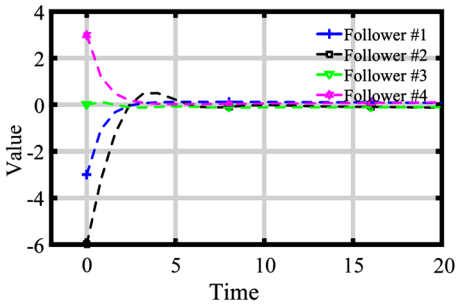

Figure 7. Distributed task data concurrency consistent execution time comparison diagram
图 7. 分布式任务数据并发一致执行时间对比图

通过图 6 和图 7 对比可以看到，第二节点数据都没有提交到 follower 节点，因为中间者检测到第二个 follower 节点未准备好，因此数据的提交操作未执行，反馈给中间者后，中间者通知 leader 节点，重新进行数据的提交操作，最后可以发现如上图 7 数据都是一致的，以此可以说明数据提交算法可以保证分布式任务并发时在数据上的协调一致。

5. 总结及展望

本文中通过 ZooKeeper 实现屏障解决分布式任务并发在时间上不能协调一致的问题，并利用文中提出的数据提交算法保证数据在分布式任务并发中的一致性，通过实验证明是可行且有用的，为解决分布式任务并发协调一致的问题提供了一种方法。

当然，文中的解决方案并不完善，效率方面也有待提高，在之后的研究和学习中将进一步完善和提高该方案的效率以及性能。

基金项目

江苏省高校自然科学研究项目“老年人行为模式识别和智能跟踪技术与应用”(18KJB520047)。

参考文献

[1] 唐宏, 刘汉江, 陈前锋, 李鹏, 等. OpenDaylight 应用指南[M]. 北京: 人民邮电出版社, 2016.

[2] 王倩雯. 基于 ODL 的分布式任务并发协调一致策略的研究与实现[D]: [硕士学位论文]. 南京: 东南大学, 2017: 6.

[3] SDNLAB.ONOS 与 OpenDaylight 比较[EB/OL]. https://blog.csdn.net/sdnlab/article/details/46547259, 2015-06-18.

[4] Running and Testing an OpenDaylight Cluster. https://wiki.opendaylight.org/view/Running_and_testing_an_OpenDaylight_Cluster

[5] zookeeper 工作原理总结[EB/OL]. https://blog.csdn.net/wangdan199112/article/details/51790005, 2016-06-30.

[6] .ZooKeeper 客户端框架 Curator 实现屏障服务(Barrier) [EB/OL]. https://blog.csdn.net/nimasike/article/details/51593358, 2016-06-06.

[7] 徐露. 分布式事务解决方案(二) [EB/OL]. https://blog.csdn.net/u010293698/article/details/52389312, 2016-08-31.

[8] 《从 PAXOS 到 ZOOKEEPER 分布式一致性原理与实践》笔记[EB/OL]. https://blog.csdn.net/paincupid/article/details/78058087, 2017-09-21.