针对 ROS 服务通信机制的软件定义改进

李冰鉴,边 境
(浙江理工大学 信息学院,浙江 杭州 310018)

摘 要:软件定义的本质是将不同硬件提供的功能抽象为标准的接口,用户通过使用抽象所得到的接口来使用具体的硬件功能从而满足不同的需求。ROS 作为一款开源的机器人通信框架,其目标是解决机器人开发过程中代码无法复用的问题。然而在其核心通信方式的服务通信机制中却存在关于服务端的单点失效问题:若提供特定服务的节点服务端失效,则服务不可用。针对该问题,提出了一种使用软件定义范式所改进的服务通信机制模型 Rosie。该模型可以在保持服务端对客户端完全透明的原有通信逻辑和优势的条件下,实现了服务通信的高可用。并且在该基础上进一步实现了对服务端的可编程管理。在设计的应用场景中,该模型对原有的服务端单点失效问题达到了预期的解决效果。更进一步,该模型在设计的过程中加入了对去中心化等特性的考虑,为未来的工作创造了必要的条件。

关键词:软件定义;ROS;服务通信;单点失效;高可用
中图分类号:TP393.0 **文献标识码**:A **文章编号**:1673-629X(2022)08-0089-07
doi:10.3969/j.issn.1673-629X.2022.08.015

Software-defined Improvement for Service Communication Mechanism of ROS

LI Bing-jian, BIAN Jing
(School of Information, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: The essence of software definition is to abstract the functions provided by different hardware into standard interfaces. Users can use specific hardware functions by using the interfaces obtained from the abstraction to meet different requirements. ROS, as an open source framework for robot communication, aims to solve the problem of code reuse during robot development. However, in the service communication mechanism of its core communication mode, there exists the problem of single point of failure on the server side: if the node providing a specific service fails, the service is unavailable. For solving this problem, Rosie, a service communication mechanism model improved by software definition paradigm, is proposed, which can realize the high availability of service communication under the condition of keeping the original communication logic and advantages of being completely transparent between the server and the client. Moreover, the programmable management of the server is further implemented. In the designed application scenario, such model achieves the expected solution to the original server single point failure problem. Furthermore, the consideration to the characteristics of decentralization in the design process is added to create necessary conditions for future work.

Key words: software-defined; ROS; service communication; SPOF; high availability

0 引 言

软件定义作为近十年来提出的新兴概念,已经出现了各种应用实现。而作为这个概念最成熟的应用之一的软件定义网络(SDN)[1-2],核心思想是将网络分为基础设施层、控制层和应用层。其中控制层通过南向接口与基础设施层通信,向交换机及路由器等基础设施下发数据包的转发规则;再通过北向接口向网络管理者提供网络相关的可编程接口,网络管理者通过使用提供的接口来编写管理程序来设定控制层,从而控制整个网络中数据包的转发行为。另一个软件定义的应用实例—软件定义存储[3-4],核心思想是将管理和存储数据的逻辑同实际存储数据的硬件相分离,从而达到使用管理一个单独存储设备的逻辑来维护由多个不同存储硬件所组成的存储集群的目的。故软件定义的本质就是将提供相同功能的不同基础设施抽象为标准的接口,然后通过使用标准接口来使用基础设施

收稿日期:2021-09-15 **修回日期**:2022-01-19
基金项目:浙江省自然科学基金(LY17F020033)
作者简介:李冰鉴(1993-),男,硕士研究生,研究方向为软件定义;通讯作者:边 境(1976-),男,硕士,副教授,研究方向为模式识别、网络安全。

的功能而忽略其在具体实现上的差异[5]。

ROS[1](Robot Operating System,机器人操作系统)是一款以 UNIX 操作系统为平台,针对机器人开发时各种功能模块不能复用的痛点所创造的分布式通信框架。经过十多年的发展和版本迭代,其几乎已经成为机器人开发领域的主流标准。ROS 主要分为四个组成部分:通信机制、开发工具、应用功能以及生态系统[6]。而其中的通信机制是建立松耦合,点对点的分布式架构的基础所在。而这其中又包括异步的话题通信,同步的服务通信以及用于公共静态数据存储和共享的参数服务器。依托于这种通信方式所支持的分布式架构,基本功能会以文件或者进程的形式存在并且共享,从而成功提高了代码复用率。但是,在服务通信机制当中却存在服务端单点失效问题:一个服务只能由一个进程注册提供,当任何提供指定服务的进程停止工作,则服务不可用。并且因为服务通信的同步特性,失效的服务可能会造成无法预测的阻塞从而导致系统整体的可用性下降。

针对以上 ROS 服务通信机制中存在的单点失效问题,该文借鉴软件定义网络的主流标准 OpenFlow[7] 通过操作流表来定义,存储和管理数据包转发规则,从而控制网络中数据包转发行为的思想。设计了一种通过“服务表”来存储服务端描述信息,然后使用手动或者自动管理机制将可用服务端通信地址提供给客户端以完成服务通信的模型 Rosie。该模型保证了在原有服务通信的逻辑上,对客户端完全透明的条件下,实现了服务通信的高可用。并且又进一步参考了版本控制系统[8]的设计思路,提供了一种针对服务表的服务端可编程管理方案。从而满足了软件定义概念中“基础设施虚拟化,管理任务可编程”[5]的基本要求。

1 背 景

1.1 软件定义

软件定义网络起源于 2006 年斯坦福大学的 Clean Slate 研究课题[9],其项目负责人 Nick McKeown 于 2008 年发表文章正式提出软件定义网络的第一个实验性协议 OpenFlow[10]。在现存的传统网络中,负责转发数据包的交换机和路由器不仅要负责数据包的转发行为,并且还要进行制定数据包向何处转发的转发决策。随着网络规模不断扩大,交换机和路由器承担的计算开销也在不断加大。为了解决网络设备日趋严重的算力紧张问题,软件定义网络利用分层的思想,将数据包的转发行为和转发决策相分离,有效降低了路由器的工作负荷。

1.2 ROS

ROS 诞生于 2007 年的斯坦福大学的 STAIR(Stanford AI Robot)项目[11],其成员发现在开发机器人软硬件系统集成过程中,将各种功能集成在一个机器人上非常困难,于是就开始尝试采用分布式的架构来连接不同的功能模块,而这也是 ROS 最终采用分布式结构的动机。2007 年 Morgan Quigley 及其导师 Andrew Y. Ng 在 AAAI 上发布了 STAIR 项目的研究成果 Switchyard[12],即 ROS 的前身。2009 年,Morgan Quigley 及其导师同一家名为 Willow Garage 的公司合作,又在 ICRA 上发表了文章[13]并向外界正式介绍了 ROS。2010 年,Willow Garage 发布了开源的 ROS 1.0 并且建立了 ROS 社区。2012 年 Willow Garage 公司又宣布成立 Open Source Robotics Foundation[14]。在 OSRF 的运营下,ROS 在近十年先后发布了若干个发行版本,截至目前最新的 ROS 版本是发行于 2020 年的 ROS Noetic Ninjemys[15]。

2 架 构

本节在以描述 ROS 系统架构以及其服务通信机制的基础上,介绍以软件定义为范式,为了解决其单点失效等问题从而重新设计的服务通信模型 Rosie。通过同现有的 ROS 框架进行对比的方式来表现 Rosie 模型所解决的问题的方式及其设计的优势所在。

2.1 整体架构

ROS 作为一种松耦合、分布式的通信框架,其中的通信单元是节点(ROS Node)。逻辑上的节点是一项功能。而实际中的节点是一个用来完成特定任务的进程或者线程。并且,只要遵循 ROS 通信机制的标准协议,每个节点可以选择不同的,受 ROS 支持的语言进行编写,并运行在不同的主机上。为了能让节点之间互相通信,ROS 提供了一个特殊的管理进程即 ROS Master,该进程是所有节点的管理者。负责提供节点信息的注册,以及服务通信中服务端的套接字地址,以供客户端查找并通信。最后还提供参数服务器的功能,用于存储和提供在所有节点之间静态的,共享的数据。ROS Master 进程在一个基于 ROS 的分布式系统中是唯一的,故必然也存在同样的单点失效问题。如果因为系统的复杂化导致管理节点的数量持续上升,那么承载节点管理服务的硬件设备也可能面临超负荷运行的风险。如图 1 所示,ROS 节点管理进程,图像获取和图像处理节点运行在主机 ROBOT 上,而图像显示节点运行在主机 LAPTOP 上。功能节点向节点管理器注册并且互相查找进行通信,从而完成特定的任务。

而在重新设计的软件定义模型 Rosie 中,包括节点管理器(Rosie Master)在内的所有功能都由节点(Rosie Node)来进行创建。这样设计对比 ROS 的优

势在于:节点管理器可以出现在系统中的任意节点上,不同系统的设计者可以根据不同情况来决定节点管理器的运行环境。除此之外该设计还为节点管理器的去中心化创造了条件。如图 2 所示,在软件定义模型 Rosie 中,图像处理节点和图像获取节点位于主机 ROBOT 上。和 ROS 不同的是,图像处理节点还同时承担了节点管理器的功能。并且,用户还可以将这个功能创建在图像获取或者图像展示等任何其他节点上。

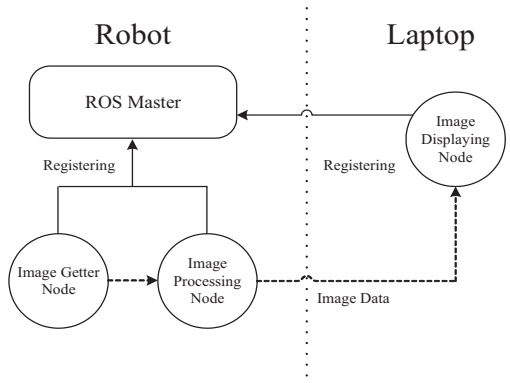

图 1　ROS 整体架构

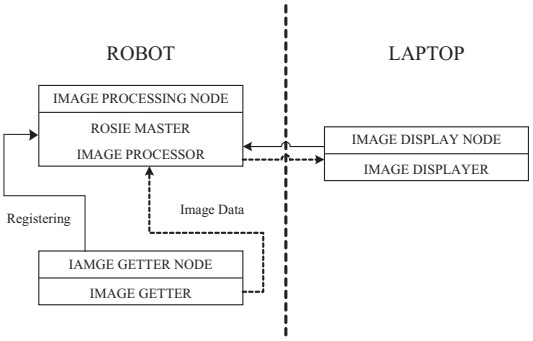

图 2　Rosie 整体架构

2.2　软件定义服务通信架构

2.2.1　服务端虚拟化

在 ROS 服务通信机制中,一个服务名称只能由一个特定的服务对象注册并提供服务功能,服务对象的相关信息通过服务名称这个服务的唯一特征注册在 ROS Master 中。需要使用该服务的客户端需要通过服务名称向 ROS Master 查询服务是否存在。若服务存在,那么 ROS Master 会将服务通信套接字地址传递给客户端,客户端通过接收到的服务地址来使用服务;若服务不存在,则需要一直等待,直到有相应服务注册。这种机制的缺点在于,一旦提供特定服务的进程失效,虽然在节点管理器中能够查找到服务端的套接字地址,却无法建立有效的连接,可能会导致系统抛出异常从而使整个系统的稳定性下降。如图 3 所示,若服务节点 ROS SERVER 停止工作,那么即使 ROS CLIENT 可以从 ROS MASTER 处查询到 ROS SERVER 的套接字地址,也无法建立连接并完成服务。

而在 Rosie 服务通信机制中,服务特征被扩充至 3 个:服务名称、输入数据类型和输出数据类型。其中服务名称决定服务逻辑,输入和输出数据类型决定该逻辑所作用的数据类型。该设计的优势在于:对于抽象的服务逻辑,可以提供多个不同的实现;而任意一种服务逻辑的实现,可以作用于不同的数据类型。相比于 ROS 中区分服务只能通过不同的服务名称,Rosie 中服务的定位方式更加有利于服务进行分类管理。

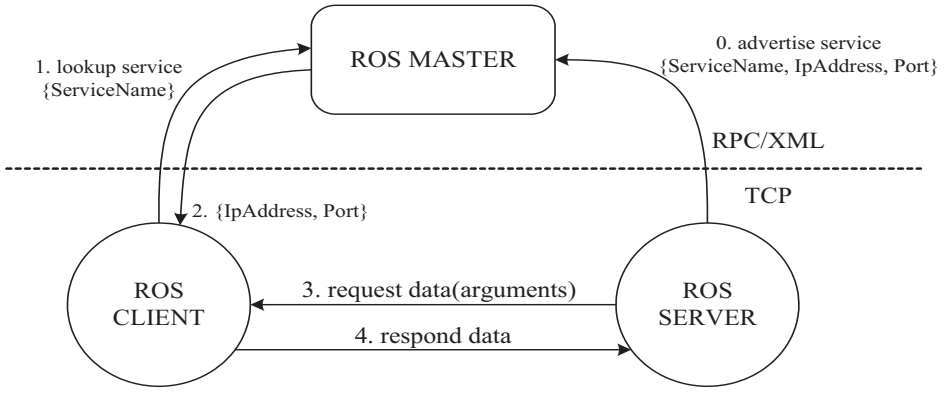

图 3　ROS 服务通信机制

如图 4 所示,Rosie 借鉴了 OpenFlow 中使用流表来管理数据包转发规则的思想,使用服务表来管理服务端信息的注册和查询。如图 5 所示,当客户端发出服务查询请求之后,Rosie Master 通过客户端提供的服务特征在服务表中查询,若服务存在,则在检查完服务有效性之后将服务信息返回给客户端;若服务不存在,则返回一个空的套接字地址对象来通知客户端其指定的服务不存在。而客户端会通过轮询的方式不断向 Rosie Master 查询该服务,直到存在为止。当客户端拿到服务端套接字地址信息之后,若成功请求服务,则完成服务流程;若套接字地址所指向的服务端失效,则重新请求 Rosie Master 查询指定的可用服务端。

最后,Rosie 并没有将服务表预设为具有特定数据结构的容器,而是将服务表设定为抽象的接口,任何实现了服务表接口的容器类所创建的容器对象都可以用来构造 Rosie Master 并作为其服务表使用。这样设计

的优势在于:不仅满足了用户和开发者可能在不同场景下自定义服务表的需求,同时也不会破坏服务通信的既定工作流程。以上设计很好地满足了软件定义概念中对于“基础设施虚拟化”的核心要求。

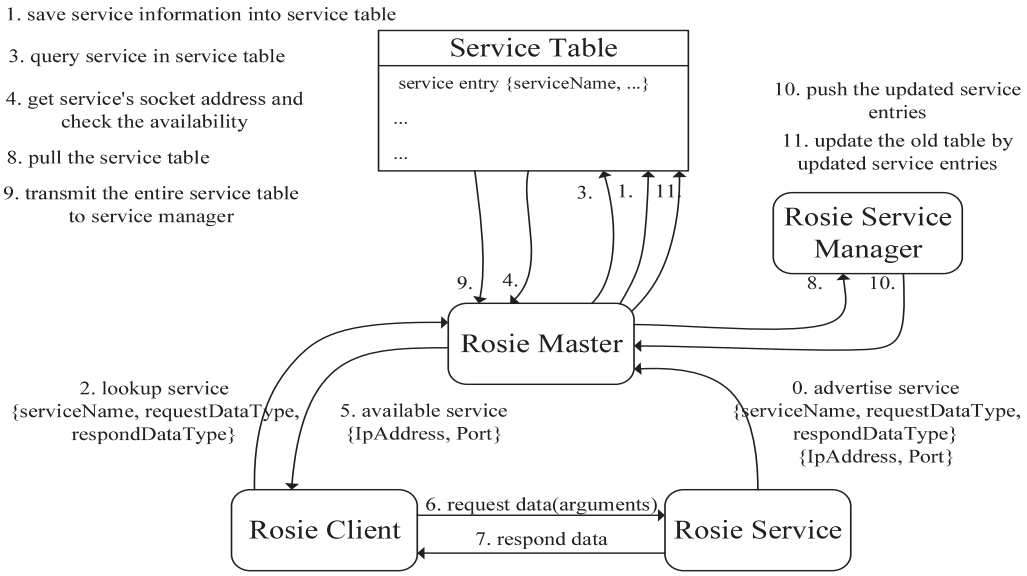

图 4 Rosie 服务通信机制

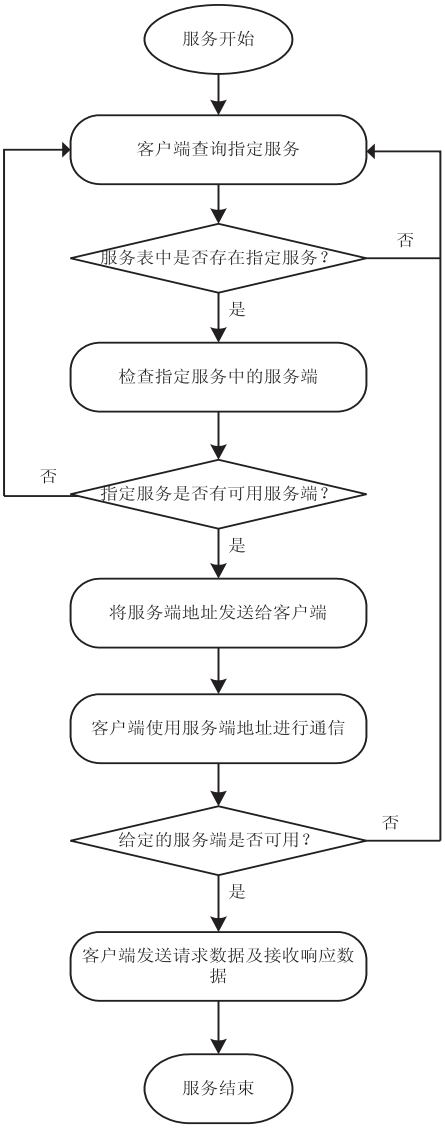

图 5 Rosie 服务通信流程

2.2.2 服务端的可编程管理

Rosie 借鉴了版本控制系统的设计思想,管理端将服务表中的内容以拉取的方式完全传输到本地,用户程序对其中的内容进行修改之后,再推送到 Rosie Master 中以更新服务表。这种管理方式的优势在于操作具有较粗的粒度,在管理过程中仅有拉取和推送两个数据传输操作,大大降低了网络开销。并且将表项数据拉取到本地之后,不仅可以利用本地算力来快速地对表进行修改,还可以检查服务表项中通过接口方法无法查看的服务特征,具有较高的扩展性。但缺点是服务管理节点可以获取服务表的全部内容,对服务表内容的操作具有完全修改权限,安全性无法得到保证。

总而言之,该管理方案基本满足了软件定义概念中对于“管理功能可编程”的核心要求。

3 实 验

3.1 方 案

在使用“基础设施虚拟化”概念来解决服务通信单点失效问题方面,实验目标任务为客户端每隔 1 秒请求一次时间服务,一共请求 10 次。提供时间服务的服务端会在启动或同客户端通信之后的 20 秒内任意时间失效。每次实验将上述任务重复 500 次。对比分别使用 ROS 和 Rosie 通信机制的客户端时间服务请求成功的次数,并且计算任务完成的成功概率。而在实现“管理功能可编程”方面,实验目标任务为在客户端请求时间服务的过程中将所有服务关闭 5 秒从而阻止客户端获得时间,然后再启用所有服务,使客户端可以继续使用时间服务。

3.2　结果与分析

当使用 ROS 服务通信机制进行时间请求任务时,若唯一的时间服务器停止工作,则时间客户端无法获得时间并完成服务。除非时间服务器重新运行,否则客户端会一直阻塞并且等待直到有可用的服务。

而使用 Rosie 服务通信机制进行相同的任务时,如图 6 所示,即使当前提供时间服务的服务器停止工作,只要系统中还有可用的时间服务器,Rosie Master 就会查询服务表并且将可用的时间服务器的通讯地址发送给客户端使其完成服务通信任务。可以看出 Rosie 服务通信机制有效解决了 ROS 中存在的单点失效问题,在原时间服务器(ID:3)失效后,可以通过新时间服务器(ID:1)来完成任务。

```
C:\Softwares\Java\JavaSEDevelopmentKit\bin\java.exe -javaagent:C:\Softwares\JetBrains\IntelliJ_IDEA_2021_1_2\lib\idea_
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"55691"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 16:27:52 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"55691"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 16:27:54 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"55691"}
RosieService/serviceId:3/serviceName:getCurrentTime/requestType:Void/respondType:String/; IS OUT OF SERVICE.
****************************************************************
[RosieManager][RosieServiceManager]serviceManagerName:ServiceManager/; starts working...
[RosieManager]Raw TableSize: 3
[RosieServiceManager]java.net.SocketTimeoutException: connect timed out
[RosieManager]Updated TableSize: 2
[RosieManager]1 unavailable services removed.
[RosieManager]Updated service table has been pushed to rosie master.
****************************************************************
private void establishServiceConnection(): java.net.SocketTimeoutException: connect timed out
SERVICE: {"serviceId":"1","serviceAddress":"192.168.152.1","servicePort":"55690"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 16:27:57 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
```

图 6　Rosie 服务通信机制克服单点失效

如表 1 所示,在 ROS 通信机制下,重复实验中每秒获取 1 次时间一共请求 10 次时间服务的任务完成率在访问触发(被访问之后会随时失效)的失效机制下只有 47.8%,服务的平均有效时长只有 9.942 秒。而在 Rosie 通信机制下,当系统中有 3 个提供相同服务的服务端在客户端需要服务时同时启动工作,在访问触发的失效机制下,服务完成率高达 96.4%,服务的平均有效时长长达 30.006 秒。即使将失效机制修改为启动触发(启动之后会随时失效),服务完成率也会高达 86.0%,平均有效时长为 15.18 秒。

表 1　不同通信机制的服务可用性

通信机制	失效策略	任务完成次数	任务完成占比/%	平均服务有效时长/s
ROS	访问触发	239	47.8	9.942
Rosie	访问触发	482	96.4	30.006
Rosie	启动触发	430	86.0	15.180

而在软件定义的另一个核心特征“管理功能可编程”的实现方面,如图 7 所示,自日志 ALL SERVICES SHUTDOWN FOR 5 SECONDS... 之前客户端最后一次接收到的时间 17:40:56 到日志 ALL SERVICE IS BACK ONLINE. 之后客户端第一次接收到的时间17:41:04 之间相差 8 秒,排除程序运行和数据传输的时间,可以认为该文采用的方案可以满足软件定义概念中“管理功能可编程”的核心要求。

最后,Rosie 中关于服务表的操作是通过调用服务表实现的接口完成,而非将其预设为某种拥有特定数据结构的容器,核心原因就是贯彻“基础设施虚拟化”的软件定义思想,使服务表可以根据不同的情况采用不同的数据结构来实现,从而实现性能上的差异化。假设系统中有 n 种不同的服务,而每种服务由 m 个服务端提供服务。公式(1)~公式(3)表示使用不同数据结构实现的服务表随着服务种类 n 和每个服务中包

含的服务端数量 m 增长时,查询指定服务的有效服务端地址的操作的时间复杂度。

```
C:\Softwares\Java\JavaSEDevelopmentKit\bin\java.exe -javaagent:C:\Softwares\JetBrains\IntelliJ_IDEA_2021_1_2\lib\idea_rt.jar=53905
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"1","serviceAddress":"192.168.152.1","servicePort":"53915"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 17:40:54 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"1","serviceAddress":"192.168.152.1","servicePort":"53915"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 17:40:56 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"1","serviceAddress":"192.168.152.1","servicePort":"53915"}
RosieService/serviceId:1/serviceName:getCurrentTime/requestType:Void/respondType:String/; IS OUT OF SERVICE.
ALL SERVICES SHUTDOWN FOR 5 SECONDS...
private void establishServiceConnection(): java.net.SocketTimeoutException: connect timed out
ALL SERVICE IS BACK ONLINE.
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"53914"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 17:41:04 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"53914"}
RosieClient: SERVICE FOUND.
RECEIVED DATA: Thu Oct 14 17:41:06 CST 2021
RosieClient: WAITING FOR AVAILABLE SERVICE...
SERVICE: {"serviceId":"3","serviceAddress":"192.168.152.1","servicePort":"53914"}
RosieService/serviceId:3/serviceName:getCurrentTime/requestType:Void/respondType:String/; IS OUT OF SERVICE.
private void establishServiceConnection(): java.net.SocketTimeoutException: connect timed out
SERVICE: {"serviceId":"2","serviceAddress":"192.168.152.1","servicePort":"53916"}
RosieClient: SERVICE FOUND.
```

图 7 Rosie 服务通信机制中可操作服务表来控制服务的可用性

若基于链表实现的服务表,其可用服务对象查询的时间复杂度为:

$$T_{\text{LinkedList}}(n,m) = n \cdot m = O(nm) \tag{1}$$

若基于使用防冲撞链的哈希表实现的服务表,其可用服务对象查询的时间复杂度为:

$$T_{\text{HashTable}}(n,m) = 1 \cdot m = O(m) \tag{2}$$

若基于搜索树实现的服务表,其可用服务对象查询的时间复杂度为:

$$T_{\text{BinarySearchTree}}(n,m) = \log_2(n) \cdot m = O(m\log_2(n)) \tag{3}$$

假设服务种类 n 与每种服务下服务端数量 m 均为线性增长,则在任意 n 与 m 的情况下,基于链表实现的服务表的最长查询耗时将是基于哈希表实现的服务表的 n 倍,是基于搜索树实现的服务表的 $\frac{n}{\log_2(n)}$ 倍。同理可得,基于搜索树实现的服务表最长查询耗时将是基于哈希表实现的服务表的 $\log_2(n)$ 倍。从以上分析可得,随着服务种类 n 的不断增长,基于不同实现的服务表在查询操作上将具有较大的性能差距。故 Rosie 通信机制允许不同用户在不同场景中使用不同实现的服务表也是设计的优势之一。

4 未来工作

虽然该模型解决了 ROS 的服务通信中所存在的单点失效问题,但在该模型中依然存在很多缺陷:

(1)虽然对服务表实现了可编程管理并且尽可能降低了可能存在的管理延迟,但是随着管理节点增长到一定数量,可能还是会因为管理时需要独占服务表而对整个系统的正常工作产生很大影响,如何解决管理节点和功能节点之间共享服务表的问题还需要深入研究。

(2)虽然模型解决了 ROS 中服务通信中服务端的单点失效问题。但是对于 ROS 整体框架来说,节点管理器 ROS Master 本身也同样存在单点失效的问题。如何对 ROS 本身进行去中心化改造是一个需要继续深入研究的问题。

(3)目前解决单点失效的问题在于系统中必须有至少一个可用的服务,若所有的服务端失效,则服务依旧不可用。那么如何使用软件定义的思想进行重新设计,将系统中可以使用的服务组合构建为已经失效的服务,使其在对客户端完全透明的条件下,满足客户端的服务请求。

5 结束语

ROS 作为一种基于 UNIX 系统的开源,分布式的通信框架,已经在机器人领域取得了令人瞩目的成就。但是其中的服务通信机制中也存在服务会单点失效的问题。针对这个问题,提出了一种以软件定义概念为

范式的改进模型 Rosie。该服务通信模型的设计参考了软件定义网络协议 OpenFlow 中使用流表来管理数据包转发行为的核心思想,通过向节点管理器中加入实现了服务表接口的容器来管理系统中的服务端。并且在此基础上进一步参考了版本管理器的设计思想,实现了通过拉取,修改和推送的方式来实现服务表内容的可编程修改,从而管理服务端可用性的设计。基本满足了“基础设施虚拟化”和“管理功能可编程”这两项软件定义概念的核心要求。在设计的实验中,该模型表现良好,基本达到了的预期目标。在文章的最后,还根据当前模型所存在的缺陷,提出了一些未来工作的可能方向。

参考文献:

[1] 张朝昆,崔　勇,唐骞祎,等. 软件定义网络(SDN)研究进展[J]. 软件学报,2015,26(1):62-81.

[2] BERA S,MISRA S,VASILAKOS A V. Software-defined networking for internet of things:a survey[J]. IEEE Internet of Things Journal,2017,4(6):1994-2008.

[3] Mware. Software-defined storage[EB/OL]. 2021. https://www. vmware. com/topics/glossary/content/software - defined-storage. html.

[4] 孙振正,龚　靖,段　勇,等. 面向下一代数据中心的软件定义存储技术研究[J]. 电信科学,2014,30(1):39-43.

[5] 梅　宏. 万物皆可互联 一切均可编程[N]. 中国信息化周报,2020-11-30(007).

[6] ROS. org. What is ROS[EB/OL]. 2018. http://wiki. ros. org/ROS/Introduction.

[7] ONF. OpenFlow specifications[EB/OL]. 2015. https://opennetworking. org/software-defined-standards/specifications.

[8] Git. About version control[EB/OL]. 2021. https://git-scm. com/book/en/v2/Getting - Started - About - Version - Control.

[9] Wikipedia. Clean slate program[EB/OL]. 2021. https://en. wikipedia. org/wiki/Clean_Slate_Program.

[10] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. OpenFlow:enabling innovation in campus networks[J]. Computer Communication Review,2008,38(2):69-74.

[11] Stanford University. STAIR[EB/OL]. 2009. http://stair. stanford. edu.

[12] QUIGLEY M, BERGER E, NG A Y. Stair:hardware and software architecture[C]//AAAI robotics workshop. [s. l.]:AAAI,2007:31-37.

[13] QUIGLEY M, GERKEY B P, CONLEY K,et al. ROS:An open-source robot operating system[C]//ICRA workshop on open source software. Hirohisa Hirukawa:[s. n.],2009.

[14] Opensource robotics foundation[EB/OL]. 2021. https://www. openrobotics. org/company.

[15] ROS. org. Distributions[EB/OL]. 2021. http://wiki. ros. org/Distributions.

(上接第 88 页)

[J]. IEEE Transactions on Communications,2019,67(6):4193-4207.

[12] HE Y,CAI Y,YU G,et al. D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks[J]. IEEE Transactions on Wireless Communications,2019,18(3):1750-1763.

[13] SALEEM U,LIU Y,LI Y,et al. Mobility aware joint task-scheduling and resource allocation for cooperative mobile edge computing[J]. IEEE Transactions on Wireless Communications,2020,20(1):360-374.

[14] YAN J,BI S,ZHANG Y. Optimal offloading and resource allocation in mobile-edge computing with inter-user task dependency[C]//IEEE global communications conference. Abu Dhabi,United Arab Emirates:IEEE,2018.

[15] BOYD S,ANDENBERGHE L V. Convex optimization[M]. Cambridge:Cambridge University Press,2004.