

1.1 研究背景与意义

随着通信技术和硬件技术的快速发展,移动设备逐渐普及。人们已经习惯了其在生活和工作中带来的便捷,移动设备成为了人们生活中不可或缺的一部分。移动设备的更新速度远远高于摩尔定律,2013年,全球移动设备已经超过 69 亿,而 2020 年则突破了百亿大关。科技的不断进步使得移动设备不仅在数量上爆炸式增长,性能也越来越强大。移动设备拥有了计算能力更加强大的处理器、更大的运行内存、更多高质量的传感器(如 GPS、光线传感器、加速度传感器、重力传感器、距离传感器等)以及更大更灵敏的高清屏幕等,已经能够媲美一些传统的计算设备,越来越多功能纷繁多样的移动应用程序(例如富媒体应用、增强现实、自然语言处理等)出现并在移动设备上执行。随着人工智能和大数据的兴起,移动应用程序的计算和数据处理越来越密集。虽然移动硬件相关技术已经取得了很大的进步,但受到重量、大小、人体工程学和散热等因素的限制,移动设备的计算资源仍然是有限的,并不能满足移动应用程序的需求。如何使用外部资源弥补移动设备资源的不足是一个重要的研究课题。

移动云计算(Mobile Cloud Computing, MCC)是解决移动设备计算资源不足问题的一般方法。在 MCC 范式下,移动设备能够将部分或全部计算任务卸载到资源丰富的云中心,从而有效扩展移动设备的计算资源。云计算是一种能够通过网络以便利的、按需付费的方式获取计算资源的范型,这些资源来自一个共享的、可配置的资源池,并能够以省力和无人干预的方式获取和释放。从服务模型的维度,云计算分为基础设施即服务(Infrastructure as a Service)、平台即服务(Platform as a Service)和软件即服务(Software as a Service)。云计算平台的基础设施通常是类似的,都是基于虚拟化技术将硬件资源进行分割或聚合,以实现按需缩减或扩展,并在虚拟化硬件之上辅以多种类型的基础软件,最终以服务的方式进行供给。随着云计算技术的普及,涌现出众多不同用途、不同类型的云计算平台,包括以 AWS

(Amazon Web Service)为代表的各种公有云服务,以及基于 CloudStack、OpenStack、Eucalyptus 等云管理系统的大量企业私有云。

但是在现有的网络环境中,MCC 范式下移动设备与云中心之间长距离数据传输导致较长响应时间的缺陷被放大,随着应用程序对时延愈发敏感,MCC 不再适用于涉及实时应用和保证服务质量的场景。移动边缘计算(Mobile Edge Computing, MEC)作为一种能更好地解决移动设备资源不足问题的高效方法应运而生,它是指在靠近物或数据源头的一侧,采用集网络、计算、存储、应用核心能力于一体的开放平台,就近提供最近端服务。其应用程序在边缘侧发起,产生更快的网络服务响应,满足行业在实时业务、应用智能、安全与隐私保护等方面的基本需求。MEC 处于物理实体和工业连接之间,或处于物理实体的顶端。作为继 MCC 之后提出的一种新型计算范式,MEC 对传统云计算的工作方式进行了革新,通过在网络边缘部署边缘服务器的方式将云中心的计算能力下沉到网络边缘,利用边缘服务器丰富的计算资源弥补移动设备的不足。

本节主要研究以下问题。首先,在云-边协同环境下,为了实现计算任务的卸载或迁移,需要一定的使能机制作为支撑;其次,虽然云计算环境下的计算资源丰富,但是合理地使用这些资源可以进一步提升用户体验以及计算资源的使用代价;最后,类似于云计算环境,在云-边协同环境下,计算资源的高效利用也是至关重要的。综上所述,本书将从面向云-边协同计算的资源管理使能机制、面向云计算的资源自适应管理方法和云边协同环境下时间驱动的工作流应用数据布局方法 3 方面展开研究,旨在更好地解决相关问题。

1.2 国内外研究现状

1.2.1 云计算

近年来,一种利用虚拟化技术的新型分布式计算模式——云计算,正在逐步发展。云计算提供无限的“云端”虚拟资源,按需共享资源池,对传统执行程序所需资源的本地供应模式是一种颠覆。基于服务类型的差异,云计算提供的服务主要可分为 3 类:软件即服务(SaaS)、平台即服务(PaaS)和基础设施即服务(IaaS)。在 SaaS 云环境下,服务提供商将软件应用程序以服务的形式传递给终端用户,而终端用户可以通过多种不同的用户接口,如 Web 浏览器,访问使用这些软件应用程序并按需付费;在 PaaS 云环境下,服务提供商提供用于搭建和开发应用程序的开发平台作为服务,包含开发语言、链接库等;而在 IaaS 云环境下,服务提供商向终端用户提供基础设施资源,如计算、存储和带宽等资源,并按需收费,本节主要针对 IaaS 云环境展开研究。另外,用户可以基于服务等级协议(Service Level Agreement, SLA),在“按需付费”的基础上使用这些云服务。SLA 协议定义用户应用必须满足的服务质量(Quality of Service, QoS)要求参数,以及用户和服务提供商各自的权利和义务。在这种背景下,研究在满足工作流应用 QoS 需求的前提下,将任务合理分配到合适资源的调度问题显得尤为重要。在云环境下,应用的复杂性和需求的多样性使得调度问题变得更加复杂,特别是针对存在复杂依赖关系的工作流调度。

云计算在复杂大规模的工作流应用处理方面具有优势。首先,云平台的“无限透明”资源可以随时随地按需提供给终端用户;其次,计算资源也可以弹性缩放地适应动态需求环境,用于执行工作流应用的资源总量可以在运行时根据执行状况做出调整;最后,IaaS 云基础设施服务让消费者根据“按需付费”模式进行资源获取和资费,克服了传统的服务的定价收费缺陷。鉴于这些云计算特色,云环境下执行工作流应用可实现更好的性能。在集成工作流环境下关于资源供应的一个关键挑战是如何确定工作流所需的执行资源数量,这些资源是虚拟机(Virtual Machine, VM)实例单元,资源总量将严重影响工作流的总执行时间和执行成本。如何采取适当的分配调度策略将资源合理分配给相应任务,并调度任务到合适的资源池,已成为云计算领域研究的主要问题。匮乏的资源配置将直接影响应用程序执行性能,而过度的资源供应会产生虚拟资源空闲情况,从而导致额外费用增加。因此,需要高效的调度算法选择最适合的资源来执行相应的任务,在满足工作流服务质量需求的同时降低执行代价。

随着云计算技术发展,许多通过工作流表示的大型应用被提交到云平台上执行。为了实现工作流间任务的协同依赖管理,同时满足不同的工作流服务质量要求,需要一种良好的调度策略为各个任务分配合适的虚拟执行资源。云环境下工作流调度框架结构图如图 1-1 所示,云计算平台上提供多种不同性能的虚拟资源,这些资源各自具有相应的配置(如 CPU、存储和 I/O 等),调度管理中心根据具体工作流的服务质量要求,利用高效的调度策略为各个工作流任务分配合适的虚拟资源。

工作流表示包含一系列执行步骤的处理过程,它简化了执行和管理应用的复杂性。工作流技术构成一种描述分布式环境下广泛科学应用的通用模型。它在许多基础科学领域的应用处理方面发挥着重要作用,如物理学、天文学、生物基因学和地震学等。工作流通常是由一个有向无环图(Directed Acyclic Graph, DAG)来表示,其中节点表示计算任务,边表示任务之间的数据或控制依赖约束。本节主要针对带截止日期约束的科学工作流展开研究。截止日期约束是工作流处理过程中的一个重要指标,例如在地震预测工作流处理过程中,及时、高效地预测地震相关情况,对人口密集地区疏散或核电站建造地点保护尤为重要。科学工作流利用分布式资源访问、管理和处理大量数据。处理和管理如此大量的数据需要一系列的计算和存储设施,然而传统的分布式环境资源往往是有限的,所以造成许多相互竞争的用户需要共享这些资源。

云环境下工作流调度问题实质是寻找一种虚拟资源与工作流任务的映射的有效方法,最优化实现工作流的预期调度目标。本节主要研究多云环境下带截止日期约束科学工作流的调度问题,如图 1-1 所示,云资源平台的资源包括多个服务提供商提供的虚拟资源总和,在考虑满足工作流任务需求的前提下,需要考虑更多调度影响因素,如多云之间数据传输带宽、多云之间的资源性质差异等。

最新的分布式计算模式,即云计算的快速发展,形成了由众多云提供商组成的高度碎片化的云市场,为处理大数据问题提供了巨大的并行计算能力。在多云中,最大的挑战之一是高效的工作流调度。虽然工作流调度问题已经得到了广泛的研究,但针对多云环境的原始工作仍然很少。

由于云服务提供商的异构性,出现了多个云服务提供商共存的“多云”局面。云计算弹性供应虚拟资源并按需付费的性质,为处理大规模科学工作流提供便利。然而,云异

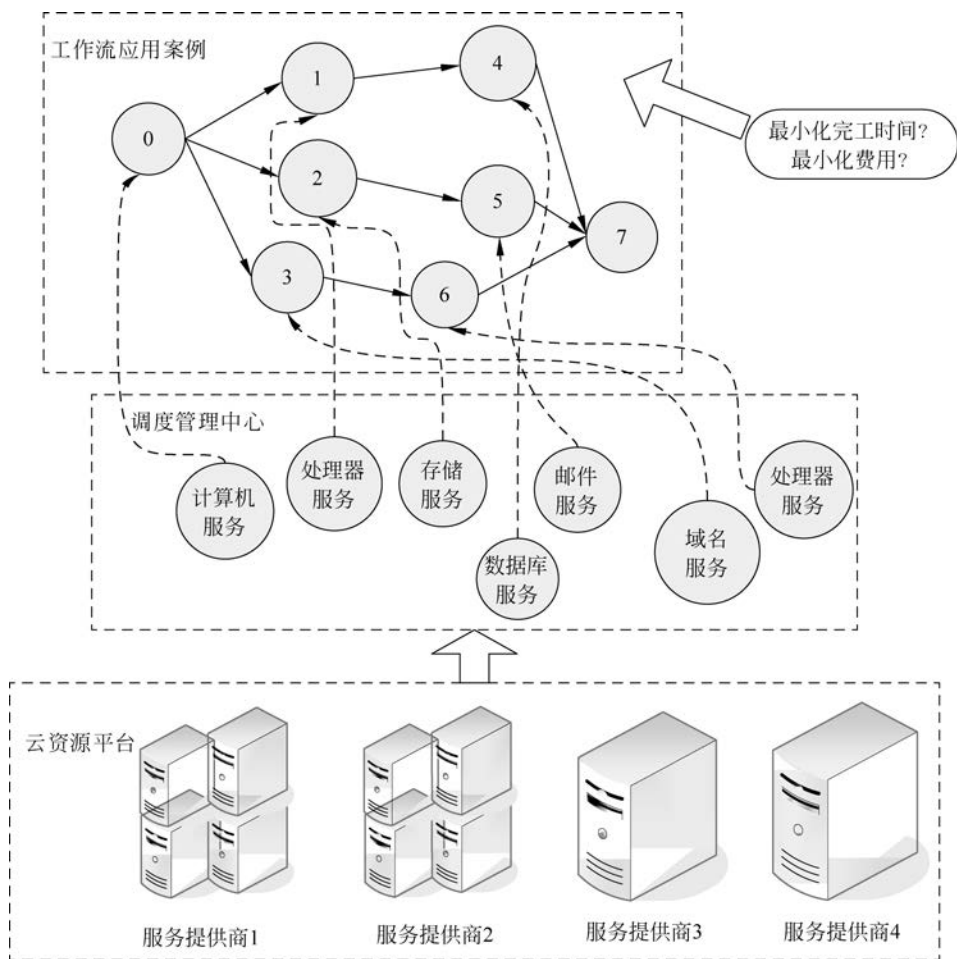


图 1-1 云环境下 workflow 调度框架结构图

构环境下的任务调度是一个 NP-hard 问题, workflow 自身的子任务之间存在复杂的时间依赖和数据依赖关系, 且多个云服务提供商之间存在诸多差异 (如要价机制、实例类型、通信带宽等), 因此, 需要一种合适的调度策略在尽可能满足 workflow 服务质量的前提下, 减少其执行代价。当前云环境下的 workflow 调度策略大多是在传统分布式计算环境 (如网格) 的 workflow 调度算法基础上做一些改进, 较少考虑云环境自身特性, 或一些调度策略仅考虑在静态单一云环境中, 单纯追求执行时间最小化目标, 未对带约束的 workflow (如截止日期) 的代价优化调度问题展开研究。

多云之间的实例类型异构、云内和云间带宽传输差异以及要价机制的多样化等因素, 使得在多云环境下调度具有依赖关联结构的带截止日期约束的科学 workflow 问题面临重大挑战。云环境下科学 workflow 调度的现有研究大多是在传统分布式计算环境 (如网格) 的基础上做相应改进, 这些工作存在云环境单一、单纯追求执行时间优化或未考虑云计算特殊性质等缺陷。云计算作为一种商业模式, 科学 workflow 调度问题不仅要考虑科学 workflow 服务质量, 还要兼顾云计算资源能耗和终端用户执行代价。因此, 如何在多云环境下, 结合云计算相关性质, 在保证科学 workflow 截止日期约束的前提下, 设计提高多云资

源总体利用率,并减少科学 workflows 执行代价的合理调度策略显得尤为迫切。

1.2.2 云-边协同计算

近年来,随着机器学习技术的不断发展,智能应用的数量迅速增加,其中深度神经网络(Deep Neural Network,DNN)在语音识别、自然语言处理和图像分析等众多领域都取得了巨大的成功。DNN 被广泛用于许多应用程序中,例如微软小娜、苹果 Siri 和谷歌 Now。但 DNN 模型过于庞大,使得它在资源有限的物联网设备上很难直接执行。这些基于 DNN 的智能应用的语音处理或图像识别比基于文本的输入需要更强大的计算处理能力,因此直接在计算资源有限的物联网设备上执行会导致精度低、时延长和能耗大等问题。所以,DNN 通常部署在计算能力强大的云服务器上。但由于云端和物联网设备之间的地理距离太远,将大量基于 DNN 的智能应用迁移到云服务器中容易导致响应时间过长和严重的网络拥塞等问题。

移动边缘计算(Mobile Edge Computing,MEC)的出现,为这类问题提供了新的解决思路。移动边缘计算通过在靠近物联网设备的网络边缘部署服务器来给用户提供更好的服务以及减轻云计算中心的负载。相比于部署在远程的云服务器上,将 DNN 部署在距离物联网设备更近的边缘服务器上具有更低的时延。而边缘服务器相比于物联网设备,又具有更强的计算能力和计算资源优势。所以,将 DNN 迁移到边缘服务器能够提高应用的执行性能,同时降低云服务器的资源开销,是解决该问题的合理有效的方式。

由于 DNN 应用数量庞大,且边缘环境网络拓扑结构复杂,所以不适当的 DNN 应用计算迁移策略可能导致计算节点之间的数据传输过多或计算节点的实际计算资源与需求的计算资源不匹配,除了给用户带来不好的用户体验外,还会导致网络拥塞等问题。因此,研究边缘环境下合理且现实的 DNN 应用计算迁移策略显得至关重要。然而,合理地利用移动设备端、边缘端和云端的资源,在边缘网络环境中部署大量 DNN 应用并不是一件容易的事情。

首先,DNN 种类繁多,不同 DNN 的结构特征大不相同。相同 DNN 之间的层次结构也相当复杂,DNN 根据功能将神经网络层划分成多个种类,这造成了层与层之间传输数据量以及不同层的计算任务复杂性相差甚多。

其次,边缘环境网络拓扑结构复杂,移动设备端、边缘端和云端的计算资源及端与端之间的网络带宽相差甚多,要综合考虑这些因素将 DNN 的各个层迁移到各个网络计算节点非常困难。

最后,服务提供商与用户之间的目标不完全相同,服务提供商希望能尽可能减少服务器运行成本并均衡各个服务器的负载,而用户希望获得好的用户体验,包括低延时、高精度等,这些不同的目标也为在边缘计算环境下进行 DNN 应用迁移决策带来了诸多困难。

DNN 在自然语言处理和计算机视觉等各种领域都非常流行。迁移策略的制定对于智能应用的使用至关重要,由于移动设备的计算资源有限,智能应用程序难以在移动设备上直接运行。许多工作专注于将 DNN 从物联网设备迁移到云服务器。Fang 等人首先设计了一种启发式方法,可以有效地调度异构服务器以进行 DNN 推理,从而满足对处

理吞吐量的要求,并保持了低响应延迟,然后,他们提出了一种深度强化学习(Reinforcement Learning, RL)方法,该方法可以最大限度地提高从学习到计划的服务质量,这项工作优化了响应延迟和推理精度,并且忽略了 DNN 中两层之间的数据传输。Qi 等人设计了一种基于 DNN 的对象检测系统,该系统具有模型调度算法,可以根据网络和移动设备的情况将 DNN 的各个层自适应地迁移到云中,这项工作还旨在减少部署基于 DNN 的应用程序时的系统延迟。Tang 等人设计了 Nanity,它是一种服务质量感知的调度方案,通过采用自适应批处理来调度 DNN 推理请求,并预先调度预测的请求以提供最佳的 GPU 资源,这不仅可以提高单个 GPU 的吞吐量,而且可以实现 GPU 资源效率。但是,仅在云服务器中部署 DNN 会造成严重的响应延迟。

现有工作大多从单个 DNN 模型角度出发,通过寻找 DNN 的分区点来使 DNN 模型可以执行在多个设备上,但是针对具体场景(例如小区、商场)中用户产生的大量 DNN 应用任务迁移决策问题还没有成熟的研究。

1.2.3 资源自适应管理

目前自适应资源管理技术主要包括基于规则驱动的自适应资源管理方法、基于控制理论的资源管理方法以及基于机器学习的资源管理方法。

基于规则驱动的自适应资源管理方法需要依赖于由专家设计的规则。Bahati 和 Bauer 提出了一种基于自适应策略驱动的自主管理系统,该系统可以确定如何最好地使用一组规则策略来满足不同的管理目标。Zhang 等人提出了一种通过利用两个自适应环路来管理 PaaS 中的基础设施资源的方法,这两个环路分别指定在中间件和虚拟机层上执行某些操作所需的规则条件。Maurer 等人考虑到工作负载变化的快速性,提出了一种用于基于规则的方法实现虚拟机自适应重配置。此外,Addis 和 Rui 等人使用启发式算法来提供高级解决方案,其基于特定领域知识搜索最合适的系统配置。但是,管理规则的制定是一个复杂的过程,需要考虑不同的软件服务特性。例如,软件服务 A 是一个实现在线系统,需要保证系统快速响应,而软件服务 B 涉及复杂计算分析,需要保证 CPU 计算资源。因此,制定规则的云工程师需要对不同软件服务进行分析,管理成本高并且管理规则复用性低。

机器学习技术使系统能够从大量的历史数据中学习出特定领域的知识。Yan 等人提出了一种基于强化学习的方法来实现云资源自我管理中的动态决策,通过设置事件触发机制,加快强化学习的收敛速度。Zheng 等人使用一个负载预测器来预测集群历史资源的利用率,并选择具有最高相似度的集群集作为训练样本到神经网络中,以实现自适应资源分配。Barrett 等人开发基于模型的 Q-学习方法,并且将 Q-学习并行化,加快代理尝试自动扩展资源的收敛速度,能够在真实的云环境中确定最佳的资源分配策略,并且输出采用该策略的原因。Alsarhan 等人将云环境中的自适应资源分配的问题定义为受约束的马尔可夫决策过程,并利用强化学习找到最大化服务提供商的增益同时保证 QoS 约束的最优策略。Xu 等人提出了一个统一的强化学习方法来自动化虚拟机和虚拟机中运行的应用的配置过程,该方法适用于云的实时自动配置,使虚拟机资源预算和设备参数设置适应云动态变化的工作负载,以提供服务质量保证。然而,基于机器学习方

法需要大量历史数据建立预测模型,在软件服务部署初期,收集的数据不足,可能会导致资源分配的误差。

1.3 主要内容和结构安排

第1章为绪论部分。本章首先介绍本作研究的面向云-边协同计算的资源管理技术的相关背景,包括研究课题的意义。然后,介绍了与本章相关的国内外研究动态及存在的问题,紧接着介绍一些课题中包含的云计算、云-边协同计算、资源自适应管理相关研究工作的进展情况,从而阐述了本书工作的必要性和重要性。最后,给出工作的研究内容及其贡献。

第2章共介绍了5种面向云-边协同计算的资源管理使能机制。对于每种机制,首先,分析了对应研究的背景意义。接着,介绍相应的问题模型。最后,在考虑不同代价的基础上提出了有效的优化调度算法。

第3章共介绍了8种面向云计算的资源自适应管理方法。对于每种方法,首先,分析了对应研究的背景意义。然后,介绍各类方法应用所需的相关概念。最后,基于云计算的基础引入有效的策略,从而提出多种面向云计算的资源自适应调度算法。

第4章共介绍了4种面向云-边协同计算的资源自适应管理方法。对于每种方法,首先,分析了工作的背景意义。接着,介绍其对应的不同场景下的问题模型。最后,提出云-边环境下基于不同的代价驱动的资源自适应管理方法。

2.1 基于运行时软件体系结构模型的混合云平台管理方法

随着云计算技术的普及,涌现出众多不同用途、不同类型的云计算平台。为了满足遗产系统整合和动态资源扩展等需求,常常需要构造混合云来统一管理不同云平台中的计算和存储资源。然而,不同云平台的管理接口和管理机制存在差异,使得开发混合云管理系统难度大、复杂度高。本节提出了一种基于运行时模型的混合云管理方法:首先,在云平台管理接口基础上,构造单一云平台的运行时模型;其次,根据云平台领域知识,提出一种云平台软件体系结构的统一模型;最后,通过模型转换,实现云平台统一模型到运行时模型的映射。于是,管理程序可以建立在云平台统一模型的基础上,降低混合云管理系统开发的难度和复杂度。本节还实现了基于运行时模型的CloudStack 和亚马逊 EC2 混合云的管理系统,并对方法的可行性和有效性进行了验证。

2.1.1 引言

云计算是一种新型计算范式,它能够通过网络为用户提供所需的计算资源。基础设施即服务、平台即服务和软件即服务是云计算常见的服务模型。然而,云计算平台的基础设施通常是类似的,都是基于虚拟化技术将硬件资源进行分割或聚合,以实现按需缩减或扩展,并在虚拟化硬件之上辅以多种类型的基础软件,最终以服务的方式进行供给。

随着云计算技术发展,云计算技术的使用得到了普及,出现了众多用途和类型不尽相同的云计算平台,例如以 AWS 为代表的公有云服务和基于 CloudStack 等云管理系统的大量企业私有云。为了满足遗产系统整合和动态资源扩展等需求,一些大型企业常常需要构造混合云来统一管理不同云平台中的计算和存储资源。然而,不同云计算

平台的管理系统存在差异,给混合云管理系统的开发带来了巨大挑战,主要来自以下两个方面:

一方面,是云平台管理接口的异构性;不同云计算平台往往提供不同类型的管理接口,在功能名称、调用方法、输入参数、返回信息及管理效果等方面均存在差异,给管理系统的开发带来了极大的复杂度。

另一方面,是云平台管理机制的异构性;不同云计算平台往往提供不同类型的管理机制,在资源分类、组织方式及系统特性等方面均存在差异,给管理系统的开发带来了极大的难度。

从系统实现的角度来看,混合云管理是一组管理任务的集合,每个管理任务由一组作用在一个或多个云计算平台上的管理操作构成,而每一个管理操作则是云平台自身提供的管理接口或是第三方提供的管理服务的调用。云计算平台的异构性导致不同云平台的管理操作存在大的差异,混合云管理系统的开发需要熟悉不同云计算平台管理操作的详细功能、调用方法和数据格式,并在此基础上编写管理程序来实现多样化的管理任务。与分析 and 决策等混合云的管理逻辑相比,管理接口调用、底层数据处理等繁杂、琐碎的编程工作并不是混合云管理的核心,但它们需要花费管理员大量的时间和精力。此外,由于管理程序建立在与特定云计算平台绑定的底层代码的基础上,其管理逻辑无法进行复用。即使管理机制类似,仍需要开发多个管理程序对不同的混合云进行管理。

混合云管理系统开发面临的主要问题是:其问题域与系统实现间存在着鸿沟,而通过硬编码实现问题域到系统实现的映射则会带来巨大的编程复杂性。软件体系结构用一组可管理的单元来表示系统的整体架构,能够作为系统需求与系统实现之间的桥梁,常用来解决需求到实现的映射过程中系统复杂性所带来的问题。因此,系统的软件体系结构建模是理解系统问题域、简化目标系统开发的一种有效方法。而当前模型驱动工程的研究也支持问题域抽象到软件实现的系统级转换。运行时软件体系结构模型用一组可管理的单元来表示系统的整体架构,通过将隐藏在系统内部的结构、状态、配置等运行时信息显式化地描述为标准的、面向管理者视角的结构化视图,能够有效地提高混合云管理系统开发的抽象层次和自动化程度。运行时软件体系结构模型已经在学术界和工业界获得了广泛的关注。大量的研究工作证明了它在不同系统与管理方式下的重要作用。

为了能够根据需求快速定制、开发混合云管理系统,本节将运行时模型引入系统开发过程中,提出一种基于运行时模型的混合云管理方法,并在实际场景中验证方法的可行性和有效性。首先,为单一的云计算平台构造运行时模型,基于管理接口实现在模型层对单一云平台进行管理;其次,根据云计算平台领域知识,提出一种云平台软件体系结构的统一模型;最后,建立统一模型与单一云平台运行时模型之间的联系,通过模型转换实现统一模型到云平台运行时模型的映射。于是,管理程序可以建立在云计算平台统一模型的基础上,极大程度地降低了开发混合云管理系统的难度和复杂度。

2.1.2 相关工作

目前,存在许多云平台管理工具,用于不同类型云资源的管理。例如,OpenStack、

Eucalyptus 用于管理基础设施层的云资源, Tivoli、Hyperic 用于管理平台层的云资源。然而, 这些管理工具缺乏完善的混合云支撑机制。

近年来, 存在许多混合云管理的研究工作, 包括混合云构造、混合云应用部署及混合云资源调度。有文献提出了一种混合云虚拟化基础设施的管理方法, 通过 OpenNebula 实现单个云平台虚拟化基础设施的管理及外部云平台核心管理 API 的集成, 并提供了一套混合云虚拟化资源的调度机制; 有文献提出了一种 IaaS (Infrastructure as a Service) 云服务的抽象模型及一组核心管理 API 的抽象接口, 并在多云环境中构造了统一管理实例, 在此基础上进一步实现了一组高级管理功能; 上述工作对混合云构造方法进行了初步探索, 然而, 方法中的统一管理接口是通过直接封装云平台核心管理 API 获得的, 因此, 其工作量大且可扩展性差。有文献在混合云虚拟化资源管理的基础上, 增加应用部署和配置功能, 实现混合云中的应用自动部署。有文献在混合云虚拟化资源管理的基础上, 面向特定应用场景研究调度策略, 实现混合云中的资源管理。本节在常见云平台共有的资源类型及管理操作的基础上, 提出了一种 IaaS 云服务的抽象模型, 通过模型转换实现多个云平台核心管理功能的快速集成, 其工作量小; 基于前期工作, 还能够支持云平台个性化管理功能的集成, 可扩展性好; 此外, 本节方法为混合云应用部署及资源调度提供了虚拟化资源的统一管理能力, 而以模型为中心的分析方法与支撑机制能够进一步为混合云管理程序的开发提供帮助。

运行时模型被广泛应用在不同类型的软件系统中, 以支持数据操作、系统自修复和动态自适应等管理功能。在前期工作中, 作者在运行时模型理论及构造方法方面进行了研究: 给定系统元模型与一组管理接口, SM@RT 工具就能自动生成代码, 在保证性能的前提下实现模型到管理接口的映射; 当系统元模型发生变化时, SM@RT 可以自动生成新的映射代码。作者还研究了系统元模型的推理方法, 通过分析调用管理 API 的客户端代码, 实现系统元模型的自动构造。同时, 为了弥补建模语言本身的非完全形式化问题, 作者在模型分析及模型容错方面也进行了研究, 提出了一种 MOF 元模型扩展机制以支持元模型的向上兼容, 从而实现在模型集成过程中模型的自动转换; 该方法在体系结构级别的系统容错实践中进一步得到了验证。作者还构建了云平台运行时体系结构模型, 对运行时模型的性能进行了验证, 并尝试基于模型语言实现系统自适应管理; 进一步, 提出一种基于模型的多样化云资源集成管理方法, 通过模型转换实现单一云平台中系统管理功能的复用和集成。本节方法建立在以上前期工作的基础上。

2.1.3 方法概览

图 2-1 是基于运行时模型的混合云管理方法的概览。该方法将运行时软件体系结构模型引入到混合云管理过程中, 通过模型转换实现云平台统一模型到单一云平台运行时模型的映射, 使得能够面向云平台统一模型进行混合云管理程序开发。该方法主要包含 3 方面工作:

- (1) 云计算平台运行时模型的构造方法;
- (2) 云平台软件体系结构的统一模型;
- (3) 统一模型到单一云平台运行时模型的映射方法。

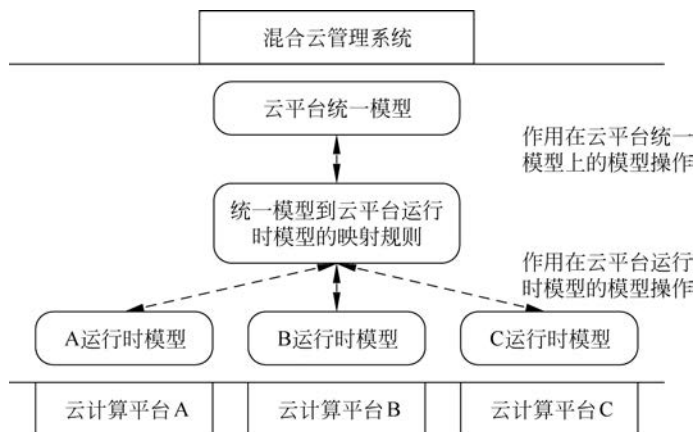


图 2-1 基于运行时模型的混合云管理方法的概览

首先,介绍一种云计算平台运行时模型的构造方法,以屏蔽云平台管理接口的异构性。云平台运行时模型是云计算平台软硬件系统的抽象。管理员仅需要对云平台受管模块的功能信息和云平台管理接口的调用方法进行描述,构造方法就能够生成相应的云平台运行时模型,并支持运行时模型与系统状态的自动同步。于是,管理员可以在模型层对云计算平台进行管理。

其次,设计一种云平台软件体系结构的统一模型,以屏蔽云平台管理机制的异构性。混合云管理实际上是对不同类型的公有云服务或企业私有云进行协同管理,然而,不同云平台在资源分类、组织方式及系统特性等方面常常存在差异。统一模型在云计算平台领域知识的基础上,对常见云平台共有的资源类型及管理功能进行描述。于是,管理员可以通过统一视图对异构云平台进行管理。

最后,提出一种模型转换方法,以实现统一模型到单一云平台运行时模型的映射。不同类型云平台的运行时软件体系结构模型存在差异,管理员仅需要对统一模型到单一云平台运行时模型的元素映射关系进行定义,转换方法就能够自动生成相应的模型转换程序,以保障模型间的同步关系。于是,管理员可以面向云平台统一模型进行混合云管理程序开发。

2.1.4 云计算平台运行时模型的构造方法

运行时模型对目标系统及其管理能力、体系结构内容与形式,以及二者之间的关系分别进行建模,将隐藏在系统内部的结构、状态、配置等运行时信息显式化地描述为标准的结构化视图。为了使管理员能够通过模型读写实现系统监控,运行时模型与运行系统需要保持因果关联:运行系统的任何时信息均将反映到运行时模型上,而运行时模型的任何变化也会作用到运行系统上。

云平台管理接口的异构性给混合云管理系统开发带来了极大的复杂度,本节使用 SM@RT 工具进行云平台运行时模型的构造,并通过运行时模型实现在模型层对单一的云计算平台进行管理。SM@RT 包含一种领域特定的建模语言和一种支持运行时系统

管理的代码生成器。开发人员使用 SM@RT 建模语言定义系统元模型和系统访问模型：

- (1) 系统元模型描述运行系统的体系结构；
- (2) 系统访问模型描述运行系统的管理能力。

在以上两种模型的基础上,SM@RT 代码生成器能够自动生成同步引擎,支持运行时模型与运行系统的双向同步。如图 2-2 所示,同步引擎为云平台中每一个虚拟机在运行时模型中创建一个可管理的“虚拟机”单元;而当某个“虚拟机”单元被删除时,同步引擎也能够发现运行时模型的变化,确定目标虚拟机并将其在云平台中关闭。

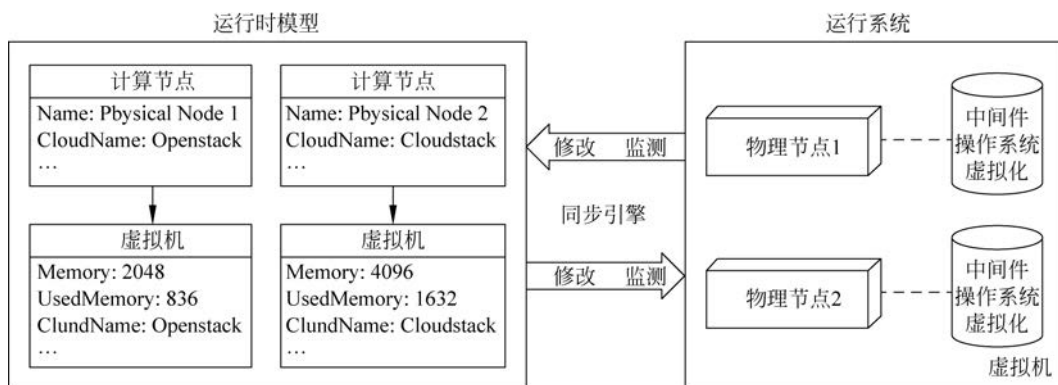


图 2-2 运行时模型与运行系统的同步

2.1.5 云平台软件体系结构的统一模型

混合云管理实际上是对不同类型的公有云服务或企业私有云进行协同管理。在云计算平台领域知识的基础上,本节提出一种云平台软件体系结构的统一模型,对常见云平台共有的资源类型及管理操作进行描述;其中,云平台的资源类型及状态信息用模型元素及其属性进行表示,而云平台的管理操作则用作用于对应模型元素的模型操作进行表示。统一模型包括使用者模型和管理员模型,分别表示面向使用者和管理员的受管资源及管理操作集合。

1. 使用者模型

使用者模型能够对云平台使用者所拥有的计算、存储、网络等资源进行统一管理,其资源分配的最小单元是服务器(Server),每个虚拟机都存在一个项目(Project)中。图 2-3 描述了使用者模型中的主要受管单元,包括账户(Account)、项目、虚拟机映像(Image)、资源配置类型(ServerType)、网络子网(Network)、磁盘卷(Volume)等。其中,Account 表示使用者账户,描述使用者的基本信息,并包含该使用者所有关联项目的列表 Projects。Project 表示项目,描述项目的基本信息,以及计算、存储、网络等资源的分配情况;项目允许多个使用者共享虚拟资源,因此,Project 包含该项目所有关联使用者的列表 ProjectAccounts。Image 表示一个虚拟机映像文件,是虚拟机软件系统的载体,Images 则表示项目可以使用的虚拟机映像文件的集合。ServerType 表示虚拟机资源配

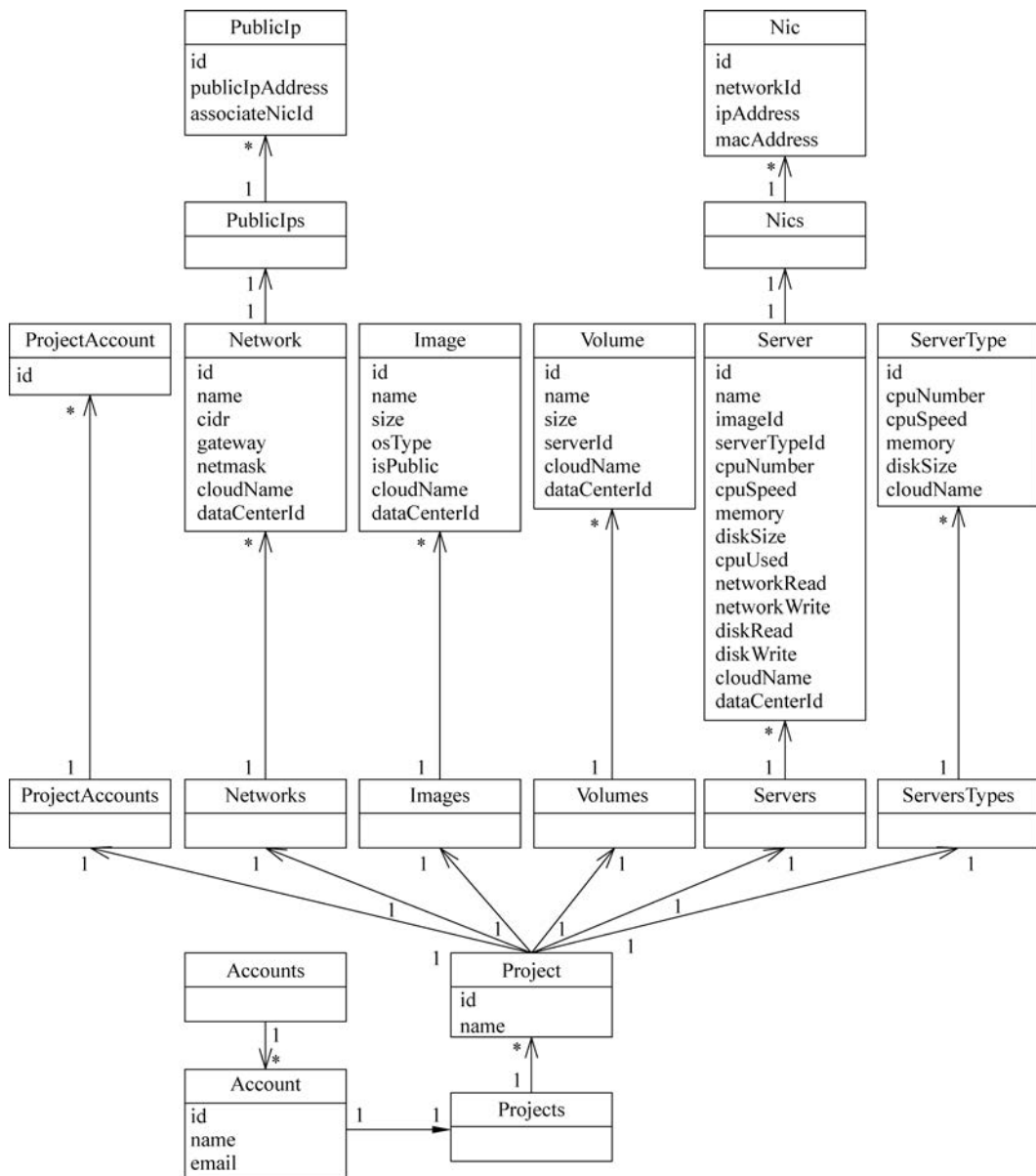


图 2-3 云平台软件体系结构的使用者模型

置类型,描述了 CPU、内存、存储等资源配置信息,ServerTypes 则表示项目可以使用的虚拟机资源配置类型的集合。Network 表示网络子网,描述该子网的基本信息,并包含该子网拥有的所有公共 IP 地址的列表 PublicIps; PublicIp 表示可供外部网络访问的 IP 地址,为网络子网中拥有该 IP 地址的虚拟机提供外部网络的访问入口。Volume 表示磁盘卷,可为虚拟机增加额外的存储,Volumes 则表示项目中所有磁盘卷的集合。Server 表示服务器,描述虚拟机的映像文件(imageId)、资源配置等基本信息,其中,资源配置信息可以通过指定虚拟机配置类型(serverTypeId)或设置 CPU、内存、磁盘等属性值两种方式进行描述;同时,Server 还包含虚拟机所使用的虚拟网卡的列表 Nics; Nic 表示虚拟

网卡,描述虚拟网卡的 IP 地址、MAC 地址,以及所在的网络子网等网络配置信息。Servers 则表示项目管理的所有虚拟机的集合。特别地,主要受管单元均包含 cloudName 和 dataCenterId 属性,分别指出该受管单元所属的云平台及数据中心。

2. 管理者模型

管理员模型能够为云平台管理提供一个的全局视图,对计算、存储、网络等资源进行统一管理。图 2-4 描述了管理员模型中的主要受管单元,包括数据中心、物理机、虚拟机映像文件、资源配置类型、存储和网络资源、项目和使用用户等。CloudManagement 是管理员模型的根元素,包含数据中心列表 DataCenters、项目列表 Projects 和使用用户列表 Accounts; 其中,Projects 表示所有项目的集合,Accounts 表示所有使用者的集合,与项目、使用者相关的受管单元在前面已有详细讨论,此处不再赘述。DataCenter 表示数据

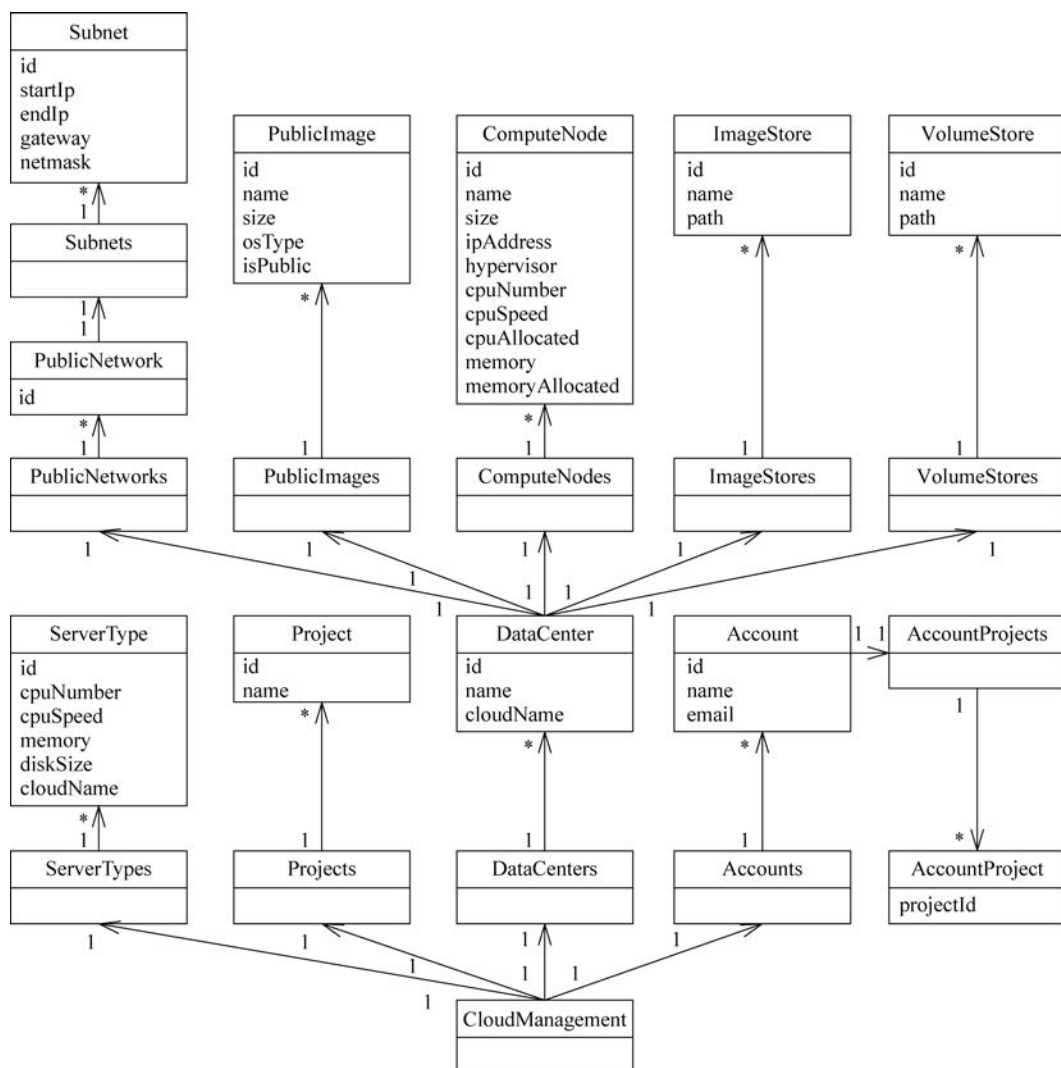


图 2-4 云平台软件体系结构的管理员模型

中心,包含物理机列表 ComputeNodes、公共虚拟机映像文件列表 PublicImages、公共虚拟机资源配置类型列表 ServerTypes、映像文件存储池列表 ImageStores、磁盘卷存储池列表 VolumeStores、网络资源列表 PublicNetworks。ComputeNode 表示安装虚拟化软件的物理节点,为虚拟机提供运行环境,描述了物理节点的计算资源等信息。PublicImage 表示公共虚拟机映像文件,可供所有项目使用,描述了映像文件的操作系统类型等基本信息。ServerType 表示公共虚拟机资源配置类型,可供所有项目使用,描述了虚拟机 CPU、内存、磁盘等配置信息。ImageStore 表示虚拟机映像文件的存储池提供了 NFS、Amazon S3 等多种形式的存储管理。VolumeStore 则表示虚拟机磁盘卷的存储池,包含 SCSI、FC-SAN 等多种形式。PublicNetwork 表示云平台拥有的公共网络资源,包含 IP 网段列表 IpRanges,描述了可供使用的公共 IP 资源。与使用者模型类似,主要受管单元也包含 cloudName 和 dataCenterId 属性,分别指出该受管单元所属的云平台及数据中心。

2.1.6 统一模型到单一云平台运行时模型的映射方法

统一模型为云计算平台的使用和管理分别提供了统一视图,为了进一步使使用者和管理员能够通过统一模型对云资源进行操作,需要实现统一模型到云平台运行时模型的映射。本节提出了一种模型转换方法,管理员仅需要定义统一模型到云平台运行时模型的映射规则,就能够通过模型操作的转换,实现模型间的同步关系。

1. 模型元素的映射关系

映射规则用于描述统一模型到云平台运行时模型的元素映射关系,任何一个统一模型中的元素属性与一个云平台运行时模型中相应的元素属性保持值的对应,且任何一组统一模型上的模型操作转换为一组云平台运行时模型上相应的模型操作以达到预期的管理效果。模型元素间存在 3 种基本映射关系,其他映射关系均可表示为以下 3 种基本映射关系的组合。

1) 模型元素间“一对一”映射关系

统一模型中的一个元素与云平台运行时模型中的一个元素对应,特别地,统一模型中元素的属性可以在云平台运行时模型中对应的元素中找到对应的属性。它们通常是指统一模型中存在一种元素,云平台运行时模型中也存在一种元素,它们均是为了描述同一类型的事物。如图 2-5 所示,统一模型中的 ServerType 与云平台运行时模型中的 InstanceType 均描述虚拟机的资源配置信息,它们是“一对一”映射关系。

2) 模型元素间“多对一”映射关系

统一模型中的两个或多个元素与云平台运行时模型中的一个元素对应,特别地,云平台运行时模型中元素的属性在统一模型中的对应属性分布在两个或多个元素中。它们通常是指统一模型中存在两种或两种以上的元素,共同描述某一事物,而云平台运行时模型中仅用一种元素来描述这一事物。如图 2-5 所示,统一模型中的 Nic 与 PublicIp 均用来描述虚拟机的网络配置信息,而云平台运行时模型中仅 Nic 用来描述其信息,它们是“多对一”映射关系。

3) 模型元素间“一对多”映射关系

统一模型中的一个元素与云平台运行时模型中的两个或多个元素对应。它们通常是指统一模型中存在一种元素,用来描述某一类事物,而云平台运行时模型中的两种或多种类型的元素分别描述该类事物的不同子类型事物。如图 2-5 所示,统一模型中的 Server 用来表示虚拟机,其属性 isHA 描述虚拟机是否具备高可用功能,而云平台运行时模型中 VirtualMachine 与 VirtualMachineHA 则均用来表示虚拟机,分别描述普通虚拟机和高可用虚拟机,它们是“一对多”映射关系。

	“一对一”映射关系	“多对一”映射关系	“一对多”映射关系
统一模型	ServerType id cpuNumber cpuSpeed memory diskSize cloudName	PublicIp id publicIpAddress associateNicId Nic id networkId ipAddress macAddress	Server id name imageId serverTypeId cpuNumber cpuSpeed memory diskSize isHA
云平台运行时模型	InstanceType id vCpus cpuSpeed memory diskSize	Nic id subnetId privateIpAddress elasticIpAddress macAddress	VirtualMachine id name imageId serverTypeId cpuNumber cpuSpeed memory diskSize VirtualMachineHA id name imageId serverTypeId cpuNumber cpuSpeed memory diskSize

图 2-5 模型元素间的 3 种映射关系

此外,统一模型中的元素属性值与云平台运行时模型中的元素属性值也存在映射关系,它们通常是指,同一种功能或内涵的配置参数或系统指标,在统一模型和云平台运行时模型中有着不同的值表示。例如,虚拟机配置信息“CPU: 2.5GHz; Memory: 1G”在统一模型中表示为 ServerTypeId 的值为“small”,在云平台运行时模型中则表示为 ServiceOffering 的值为“a7d50774-d553-4ea6-b03e-6a168af2de14”;因此,在统一模型到云平台运行时模型的映射过程中,也需要保持其值的对应关系。

2. 模型操作的映射规则

云平台使用者和管理员通过模型操作进行管理任务的执行。云平台的管理任务,本质上是云资源的增减及其属性的查改。分别对应模型元素的 List、Add 和 Remove 操作及其属性的 Get 和 Set 操作。为了实现统一模型到云平台运行时模型的映射,需要将任何一个统一模型上的模型操作转换为一个云平台运行时模型上对应的模型操作,以达到预期的管理效果。如表 2-1 所示,定义了模型操作的映射规则:

表 2-1 模型操作的映射规则

	“一对一”映射规则	“多对一”映射规则	“一对多”映射规则
示例	$A \rightarrow B$ $A1, a1 \rightarrow B1, b1$	$A \rightarrow B$ $A1, a1 \rightarrow B1, b1$ $C1, c1 \rightarrow B1, b2$	$A \rightarrow B$ 或 $A \rightarrow C$ $A1, a1 \rightarrow B1, b1$ $A2, a1 \rightarrow C1, c1$

续表

	“一对一”映射规则	“多对一”映射规则	“一对多”映射规则
Get	Get A1. a1 $\rightarrow$ Get B1. b1	Get A1. a1 $\rightarrow$ Get B1. b1 Get C1. c1 $\rightarrow$ Get B1. b2	Get A1. a1 $\rightarrow$ Get B1. b1 Get A2. a1 $\rightarrow$ Get C1. c1
Set	Set A1. a1 $\rightarrow$ Set B1. b1	Set A1. a1 $\rightarrow$ Set B1. b1 Set C1. c1 $\rightarrow$ Set B1. b2	Set A1. a1 $\rightarrow$ Set B1. b1 Set A2. a1 $\rightarrow$ Set C1. c1
List	List * A $\rightarrow$ List * B Get A. properties $\rightarrow$ Get B. properties	List * A $\rightarrow$ List * B Get A. properties $\rightarrow$ Get B. properties	List * A $\rightarrow$ List * B 和 List * A $\rightarrow$ List * C Get A. properties $\rightarrow$ Get B. properties 或 Get A. properties $\rightarrow$ Get C. properties
Add	Add * A $\rightarrow$ Add * B Set A. properties $\rightarrow$ Set B. properties	Add * A $\rightarrow$ Add * B Set A. properties $\rightarrow$ Set C. properties 和 Set A. properties $\rightarrow$ Set B. properties	Add * A $\rightarrow$ Add * B 或 Add * A $\rightarrow$ Add * C Set A. properties $\rightarrow$ Set B. properties or Set C. properties
Remove	Remove * A $\rightarrow$ Remove * B	Remove * A $\rightarrow$ Remove * B	Remove * A $\rightarrow$ Remove * B 或 Remove * A $\rightarrow$ Remove * C

1) 模型元素间“一对一”映射规则

存在统一模型中元素 A 与云平台运行时模型中元素 B 对应；那么，对元素 A 的 add、remove 或 list 操作转换成对元素 B 的相同操作，对元素 A 属性的 get 或 set 操作则转换成对元素 B 对应属性的相同操作。

2) 模型元素间“多对一”映射规则

存在统一模型中的元素 A 与云平台运行时模型中元素 B 对应，且元素 B 的某些属性同时与统一模型中 C 元素的属性关联；那么，对元素 A 和元素 C 属性的 get 或 set 操作转换成对元素 B 对应属性的 get 或 set 操作，对元素 A 的 add、remove 或者 list 操作则转化成对元素 B 的相同操作，特别地，当元素 B 被创建时，属性的初始值同时从元素 A 和元素 C 中获取。

3) 模型元素间“一对多”映射规则

存在统一模型中的元素 A 与云平台运行时模型中元素 B 或元素 C 对应；那么，对元素 A 及其属性的操作转化成对其对应元素和属性的相同操作；特别地，对元素 A 的 list 操作则转换成同时对元素 B 和元素 C 执行 List 操作。

3. 模型操作的描述方式

通过模型操作的转换，任何作用在统一模型上的操作将映射为作用在云平台运行时模型上的对应操作。如图 2-6 所示，这些模型操作将以 XML 文件的形式进行描述，同时，根据云平台的基本信息，传送到目标云平台运行时模型并执行，以实现预期的管理效果。

List	描述	<pre> <action node="TypeS" type="List"> <query node="TypeF" condition="Constraint" /> </action> <return> <node="TypeS" condition="Constraint1" /> <node="TypeS" condition="Constraint2" /> ... </return> </pre>
	Pre-前提条件	$\exists \text{TypeS } s, \exists \text{TypeF } f, s \in f \wedge f \text{ in condition of Constraint}$
	示例	Find the "TypeF" element which satisfies the constraints and list "TypeS" elements which are its child nodes.
Get	描述	<pre> <action key="KEY" type="get"> <query node="Type" condition="Constraint" /> </action> <return key="KEY" value="VALUE" /> </pre>
	Pre-前提条件	$\exists \text{Type } n, n \text{ in condition of Constraint} \wedge \text{"KEY"} \in \text{Type.attributes}$
	示例	Find the "Type" element which satisfies the constraints and get the value of its "KEY" attribute.
Add	描述	<pre> <action node="TypeS" type="add"> <query node="TypeF" condition="Constraint" /> <set key="KEY1" value="VALUE1" /> <set key="KEY2" value="VALUE2" /> </action> </pre>
	Pre-前提条件	$\exists \text{TypeF } f, f \text{ in condition of Constraint} \wedge \text{"KEY1", "KEY2"} \in \text{TypeS.attributes}$
	示例	Find the "TypeF" element which satisfies the constraints and add a "TypeS" element as its child node.
Set	描述	<pre> <action key="KEY" value="VALUE" type="set"> <query node="Type" condition="Constraint" /> </action> </pre>
	Pre-前提条件	$\exists \text{Type } n, n \text{ in condition of Constraint} \wedge \text{"KEY"} \in \text{Type.attributes}$
	示例	Find the "Type" element which satisfies the constraints and set the value of its "KEY" attribute to "VALUE".
Remove	描述	<pre><action node="Type" condition="Constraint" type="remove" /></pre>
	Pre-前提条件	$\exists \text{Type } n, n \text{ in condition of Constraint}$
	示例	Find the "Type" element which satisfies the constraints and remove it.

图 2-6 5 种模型操作

2.1.7 实验与评估

为了满足遗留系统整合和动态资源扩展等需求,一些大型企业常常需要构造混合云来统一管理不同云平台中的计算和存储资源。然而,目前尚不存在针对混合云管理的成熟的开源解决方案。为了验证本节方法的可行性和有效性,针对 Amazon EC2 公有云服务和基于 CloudStack 的企业私有云,从云平台使用者的角度,构造混合云管理系统,实现了面向云平台统一模型的管理程序开发。

1. CloudStack 与 Amazon EC2 运行时模型的构造

CloudStack 是一种开源的云平台管理软件,常用于企业私有云管理。图 2-7 模型图(上)描述了 CloudStack 软件体系结构模型(系统元模型)中的主要受管单元,包括项目(Project)、计算配置方案(ServiceOffering)、磁盘配置方案(DiskOffering)、虚拟机映像文件(Template)、虚拟机(VirtualMachine)和磁盘卷(Volume)等资源。其中,Project 表示项目,描述了项目的基本信息,包含计算配置方案列表 ServiceOfferings、磁盘配置方案列表 DiskOfferings、虚拟机映像文件列表 Templates、虚拟机列表 VirtualMachines、网络子网列表 GuestNetworks 和磁盘卷列表 Volumes。Template 表示项目可以使用的虚拟机映像文件,描述其操作系统信息。ServiceOffering 表示虚拟机计算配置方案,描述其 CPU 核数、内存大小等计算资源配置信息。DiskOffering 表示虚拟机磁盘配置方案,描述其磁盘大小等存储资源配置信息。GuestNetwork 表示网络子网,描述该子网的基本信息,并包含该子网拥有的公共 IP 地址的列表 PublicIps; PublicIp 表示可供外部网络访问的 IP 地址,为子网中拥有该 IP 地址的虚拟机提供外部网络的访问入口。Volume 表示磁盘卷,可为虚拟机增加额外的存储。VirtualMachine 表示虚拟机,描述虚拟机使用的映像文件、计算及存储配置方案等基本信息,同时,VirtualMachine 还包含虚拟机使用的虚拟网卡列表 Nics; Nic 表示虚拟网卡,描述其 IP 地址、MAC 地址及所在的网络子网等网络配置信息。Image 表示虚拟机映像文件,是虚拟机软件系统的载体。Subnet 表示某个网络子网,为虚拟机提供基本的网络服务。Volume 表示可定制的用于持久性数据存储的磁盘卷,可为虚拟机增加额外的存储空间。Instance 表示虚拟机,描述了虚拟机使用的映像文件、资源配置类型等基本信息,同时,Instance 包含其使用的虚拟网卡列表 Nics; Nic 表示虚拟网卡,描述虚拟网卡的 IP 地址、MAC 地址及所在的网络子网等网络配置信息。

给定 CloudStack 和 Amazon EC2 系统元模型,仍需要定义其上的模型操作,即访问模型; CloudStack 和 Amazon EC2 拥有大量的管理接口,通过定义模型操作到这些管理接口的映射规则,来对它们建模。在系统元模型和访问模型的基础上,SM@RT 工具能够自动生成模型转换程序,以保障系统运行时模型与运行系统的双向同步。于是,能够在模型层对 CloudStack 和 Amazon EC2 进行使用和管理。

2. 统一模型到云平台运行时模型的映射

如图 2-7 所示,统一模型(使用者模型)为云资源的使用提供了统一的全局视图;为了进一步使使用者能够通过统一模型对基于 CloudStack 的企业私有云及 Amazon EC2 公有云服务的计算、存储资源进行操作,需要实现统一模型到 CloudStack 和 Amazon EC2 运行时模型的映射。根据模型间的元素映射关系,定义了统一模型到云平台运行时模型的映射规则;其中,云平台虚拟机及网卡在模型间的映射关系,是映射规则描述的关键点。下面以统一模型中 Server 到 CloudStack 运行时模型中 VirtualMachine 的“一对一”映射,以及统一模型中 Nic 和 PublicIp 到 CloudStack 运行时模型中 Nic 的“多对一”映射为例,详细介绍映射规则的描述方式。

统一模型中 Server 与 CloudStack 运行时模型中 VirtualMachine 均表示虚拟机,它

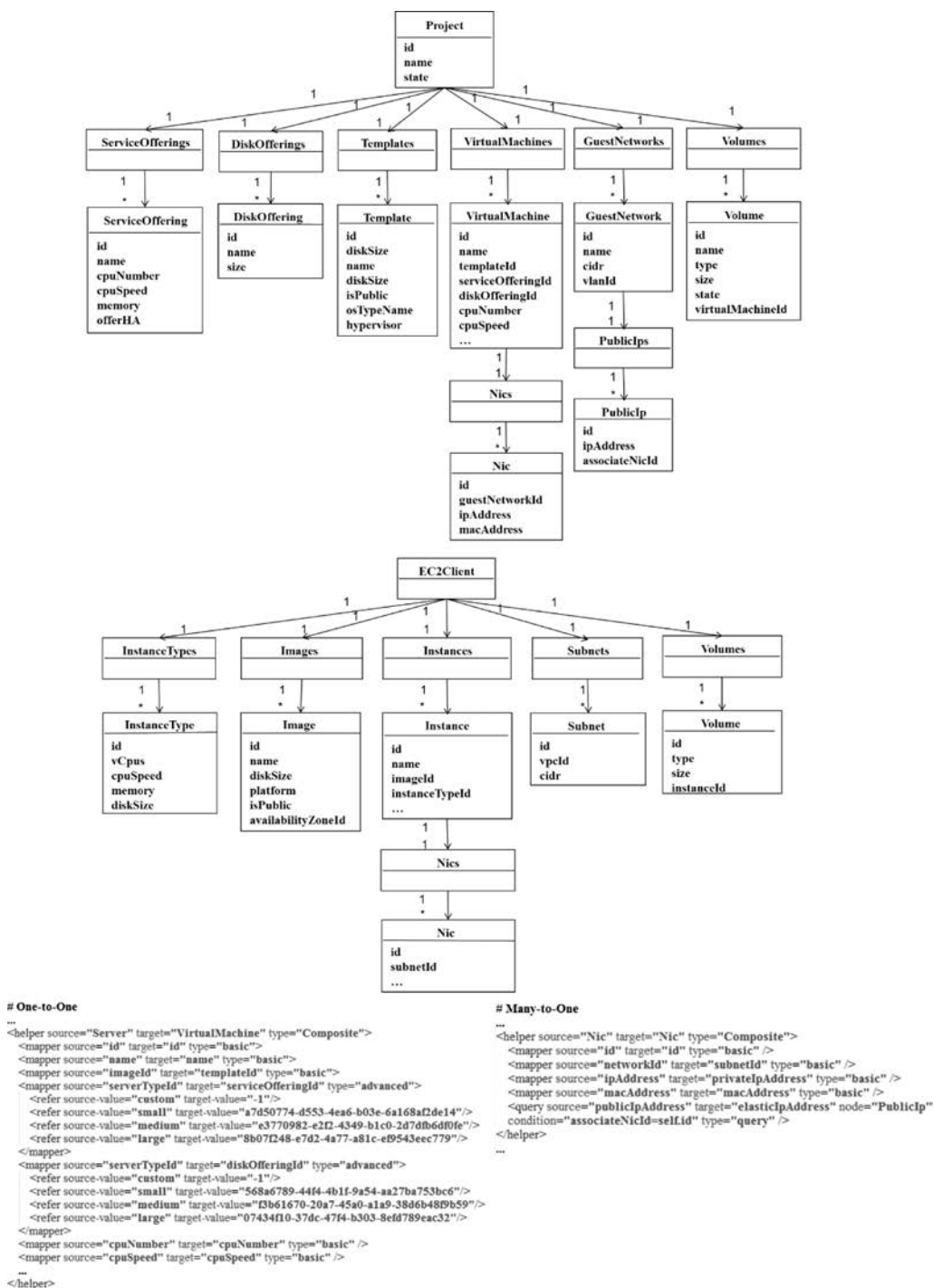


图 2-7 CloudStack 模型(上)、Amazon EC2 模型(下)及模型元素间的映射关系

们存在“一对一”映射关系,在如图 2-7 所示的映射规则描述片段中通过 helper 标签表示两个元素间的映射关系。Server 中 id、name、imageId、cpuNumber、cpuSpeed、cpuUsed 及 diskSize 等属性与 VirtualMachine 中 id、name、templateId、cpuNumber、cpuSpeed、