软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(7):1476−1491 [doi: 10.13328/j.cnki.jos.004457]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于运行时模型的多样化云资源管理方法*

陈 星[1,2,3], 张 颖[1,2], 张晓东[1,2], 武义涵[1,2], 黄 罡[1,2], 梅 宏[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](福州大学 数学与计算机科学学院,福建 福州 350108)

通讯作者: 黄罡, E-mail: hg@pku.edu.cn

摘 要: 多样化的受管资源和不断变化的管理需求,使得云管理面临很大的难度和复杂度.面对一个新的特定的管理需求,管理员往往是在已有管理软件的基础上进行二次开发,通过管理功能的获取和组织来构造新的管理系统.然而,由于缺乏通用的方法,二次开发的难度和复杂度依然很大.为了能够根据管理需求快速定制、集成、扩展已有的管理软件,提出一种基于运行时模型的多样化云资源管理方法.首先,在系统管理接口的基础上构造不同受管资源的运行时模型;其次,通过对不同的运行时模型进行合并,来构造包含所有目标受管资源的组合模型;最后,通过组合模型到用户特定模型的转换,来满足特定的管理需求.在 OpenStack 与 Hyperic 两款独立管理软件的基础上,实现了基于运行时模型的虚拟机软、硬件资源统一管理系统,验证了方法的可行性和有效性.

关键词: 云管理;软件体系结构;运行时模型

中图法分类号: TP311

中文引用格式: 陈星,张颖,张晓东,武义涵,黄罡,梅宏.基于运行时模型的多样化云资源管理方法.软件学报,2014,25(7):1476−1491. http://www.jos.org.cn/1000-9825/4457.htm

英文引用格式: Chen X, Zhang Y, Zhang XD, Wu YH, Huang G, Mei H. Runtime model based approach to managing diverse cloud resources. Ruan Jian Xue Bao/Journal of Software, 2014,25(7):1476−1491 (in Chinese). http://www.jos.org.cn/1000-9825/4457.htm

Runtime Model Based Approach to Managing Diverse Cloud Resources

CHEN Xing[1,2,3], ZHANG Ying[1,2], ZHANG Xiao-Dong[1,2], WU Yi-Han[1,2], HUANG Gang[1,2], MEI Hong[1,2]

[1](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

[3](College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)

Corresponding author: HUANG Gang, E-mail: hg@pku.edu.cn

Abstract: Due to the diversity of resources and different management requirements, cloud management is faced with great challenges in complexity and difficulty. For constructing a management system to satisfy a specific management requirement, redeveloping a solution based on existing management system is usually more practicable than developing the system from scratch. However, the difficulty and workload of redevelopment are also very high. In this paper, a runtime model based approach is presented to managing diverse cloud resources. First, the runtime model is constructed for each type of cloud resources based on their management interfaces. Second, the composite runtime model is build for all managed resources through merging their runtime models. Third, cloud management is setup to meet specific requirements through model transformation from the composite model to the customized models. Additionally, based on OpenStack and Hyperic, a runtime model based management system is implemented to manage the hardware and software resources of virtual machines with the proposed approach. The results prove that new approach is feasible and effective.

* 基金项目: 国家自然科学基金(61222203, 60933003, 61121063); 国家高技术研究发展计划(863)(2013AA01A208, 2012AA010107); 中国博士后科学基金(2013M530011)

收稿时间: 2013-01-08; 修改时间: 2013-04-27; 定稿时间: 2013-06-27

Key words: cloud management; software architecture; runtime model

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[1].虽然云计算按照服务模型可以分为基础设施即服务(IaaS)、平台即服务(PaaS)和软件即服务(SaaS)这 3 种,但是以上这 3 种不同类型的云的基础设施却通常类似:它们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.云计算的快速发展对其管理带来了挑战,主要来自以下两个方面:

- 一方面是多样化的受管资源.云中存在大量不同类型的软硬件资源,包括 CPU、内存、网络、存储等硬件资源,也包括 Web 服务器、应用服务器、数据库服务器等软件系统.同一种资源往往会有多种不同的管理软件,例如,虚拟化软件就包括 XEN,KVM,VMware 等.而不同的管理软件还可能提供多种类型的管理接口,例如 API、执行脚本、配置文件和 Web 控制台等;
- 另一方面是不断变化的管理需求.管理需求包括目标受管系统以及与管理员知识、经验相符的用户偏好.受管系统的不断变化,导致管理软件常常需要增加新的管理功能,来对新增资源进行管理;管理员知识背景和管理经验的差异性,导致管理软件常常需要以不同的方式对管理功能进行组织,来满足用户偏好的不断变化.

从系统实现的角度来看,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种或多种云资源上的管理操作构成,而每一个管理操作则是云资源自身提供的管理接口或者第三方提供的管理服务的调用.云计算服务模型和部署模型的差异性,导致不同的云的管理任务往往各不相同.例如,Amazon EC2 主要对虚拟机等基础设施级别的云资源进行管理,而 Google App Engine 则主要对操作系统、系统运行环境等平台级别的云资源进行管理.学术界和产业界都对云管理进行了深入的研究,存在大量的云管理软件,它们针对不同的云资源,在其管理接口或第三方提供的管理服务的基础上,实现了一组特定的管理任务.面对一个新的特定的管理需求,管理员往往是在已有管理软件的基础上进行二次开发,通过管理功能的获取和组织来构造新的管理系统.然而,多样化的受管资源和不断变化的管理需求,给二次开发带来了很高的复杂度和难度:一方面,多样化的受管资源给管理功能的获取带来了很高的复杂度,管理员不仅需要熟悉不同类型的管理软件,还需要能够有效使用不同类型的管理接口,以实现管理功能的交互和数据的转换;另一方面,不断变化的管理需求给管理功能的组织带来了很高的难度,管理员需要根据特定的受管资源和用户偏好,在资源管理接口的基础上构造管理系统.

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,能够扮演系统需求与系统实现之间的桥梁[2],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[3].此外,模型驱动工程的研究也已经能够支持问题域抽象描述到软件具体实现的转换[4].因此,用体系结构模型来描述多样化的云资源,并实现其与运行系统的双向转换,可以有效减小管理需求与系统实现间的鸿沟.

为了能够根据管理需求快速定制、集成、扩展已有的管理软件,以实现多样化云资源的管理,本文将运行时模型引入到云管理过程中,建立了基于运行时模型的多样化云资源的管理方法,并在实际场景中验证方法的可行性和有效性.本文包含 3 个主要工作:

1) 提出一种云资源运行时模型的构造方法,该方法基于已有管理软件的管理接口实现了在模型层对单一受管资源进行管理;
2) 提出一种分布式运行时模型的合并方法,该方法在运行时模型的基础上实现了通过组合模型对多种受管资源进行统一的管理;
3) 提出一种组合模型到用户特定模型的转换方法,该方法通过建立模型间的元素映射关系来满足特定的用户偏好.

本文第 1 节概述方法的整体框架.第 2 节介绍云资源运行时模型的构造方法.第 3 节介绍分布式运行时模型的合并方法.第 4 节介绍组合模型到用户特定模型的转换方法.第 5 节介绍实例研究并对方法的可行性和有

效性进行评估.第 6 节与相关工作进行比较.最后一节总结全文并展望将来的工作.

1 方法概览

图 1 是基于运行时模型的多样化云资源管理方法的概览.

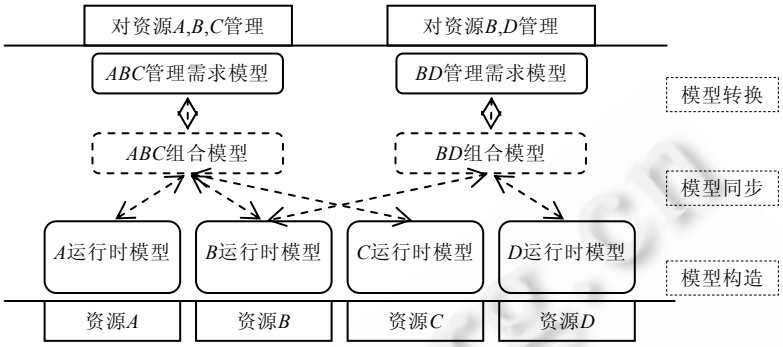


Fig.1 Overview of the runtime model based approach to managing diverse cloud resources

图 1 基于运行时模型的多样化云资源管理方法概览

方法将运行时体系结构模型引入到云资源管理过程中,通过模型构造、模型合并和模型转换来实现满足用户特定需求的管理系统.该方法主要包含 3 方面工作:

1) 云资源运行时模型的构造;
2) 分布式运行时模型的合并;
3) 组合模型到用户特定模型的转换.

首先,提出一种云资源运行时模型的构造方法,以屏蔽云资源管理接口的异构性.云资源运行时模型是云环境中软硬件系统的抽象.管理员仅需对系统受管模块的信息和系统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资源运行时模型,并支持运行时模型与系统状态的自动同步.于是,管理员就可以在模型层对云资源进行管理.

其次,提出一种分布式运行时模型的合并方法,以屏蔽云资源的分布性.云管理实际上是云资源的管理,而不同管理需求的受管资源则不尽相同.管理员仅需根据特定管理需求,在云资源运行时模型基础上,选择代表其受管资源的模型片段,合并方法就能够自动生成相应的组合模型,并支持组合模型与单个运行时模型的数据同步.于是,管理员就可以通过组合模型实现多种云资源的统一管理.

最后,提出一种组合模型到用户特定模型的转换方法,以简化管理系统的实现.管理系统提供给不同类型的管理员使用,而其知识背景和管理经验等则不尽相同.管理员仅需根据用户偏好构造用户特定模型,并对模型间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,以支持组合模型和用户特定模型的双向同步.于是,管理员就可以用偏好的方式对云资源进行管理.

2 云资源运行时模型的构造方法

云资源运行时模型是云环境中实际软硬件系统的抽象,其构造方法的输入包括系统的元模型和访问模型,其中,元模型描述了受管系统的信息,访问模型描述了管理接口的调用方法.基于以上输入,目标系统运行时体系结构能够由 SM@RT 工具[5]自动生成,而它与系统间的双向一致性也能够得到保证.

如图 2 所示,同步引擎检测到运行环境中的 JonAS,并自动地在运行时模型中添加一个 JOnAS 元素;当管理员在运行时模型中删除这个 JOnAS 元素时,同步引擎也能够自动检测到这个变化,并调用管理接口关闭相应的 JOnAS.我们在之前的工作[5−7]中对运行时模型的构造方法与支撑机制进行了详细的介绍,这里不再赘述.

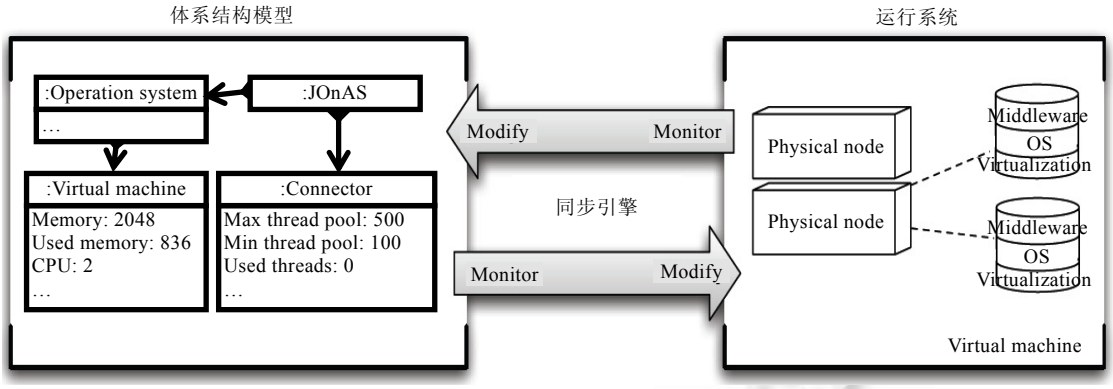


Fig.2 Synchronization between the runtime model and the running system

图 2 运行时模型与运行系统的同步

3 分布式运行时模型的合并方法

云管理实际上是云资源的管理,而不同管理需求的目标受管资源也各不相同.因此,我们提出了分布式运行时模型的合并方法,在云资源运行时模型的基础上,以组合模型的形式对其进行重组织,实现了多种云资源的统一管理,主要工作包括模型定制和数据同步.

3.1 模型定制

模型定制是指针对特定的管理需求,根据其目标受管资源,在多个云资源运行时模型中截取相应的模型片段,将它们组合成一个新的模型,即组合模型.

在图 3 所示场景中,需要对 IP 为“10.10.1.3”的虚拟运行环境进行管理,包括虚拟机的虚拟硬件资源及其上基础软件 JOnAS 的管理,以期望实现虚拟运行环境的资源优化配置.虽然现有环境中存在一个虚拟机管理系统对环境中的所有虚拟机进行管理,以及一个应用服务器管理系统对所有应用服务器进行管理,但以上两个系统均不能同时对虚拟机和应用服务器进行管理.因此,我们在以上两个系统的运行时体系模型基础上进行模型的定制,截取虚拟机管理系统运行时模型中的 IP 为“10.10.1.3”的虚拟机元素和应用服务器管理系统运行时模型中的 IP 为“10.10.1.3”的 JOnAS 元素,将它们组合为一个新的特定于该管理需求的模型.特别地,组合模型不需要包含模型元素的全部属性,而可以仅包含其中的某几个属性.例如,图 3 所示场景中的组合模型可以只包含 JOnAS 的线程池、连接池属性以及虚拟机的 CPU、内存属性.

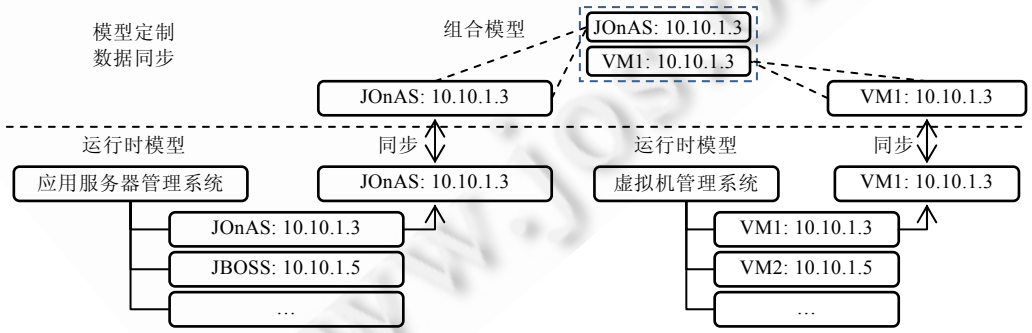


Fig.3 An example of merging distributed runtime models

图 3 分布式运行时模型合并的例子

定制的模型片段需要能够从云资源运行时模型中获取受管资源信息,即模型片段的识别.云资源运行时模型与用户定制的模型片段均是以 XML 文件的形式进行存储,对于模型中的每一个元素,都有且仅有一条从根节

点出发的路径可以定位到该元素.如图 4 所示,用户定制的模型片段(右图)和相应的运行时模型(左图)有着类似的组织结构.用户所定制的每一个元素,都可以从运行时体系结构的根节点开始,根据定制的路径一层一层地找到相对应的元素,从而实现整个模型片段和运行时模型的数据关联.例如,“vid”为 3 的 Platform 的 CPU 参数值在用户定制的模型片段中,可以通过路径“⟨PaaSList⟩→⟨PaaS id=2⟩→⟨Platform vid=3⟩→CPU”找到,而在运行时模型中的查询路径则也是类似的.

系统运行时模型

```
<?xml version="1.0" encoding="UTF-8"?>
<SCloud:Cloud xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/
  <paaSList>
    <paaS id="1">
      <platform xsi:type="SCloud:JOnAS" VID="1" Storage="25" CPU="12" Memory="50" />
      <platform xsi:type="SCloud:JOnAS" VID="2" Storage="55" CPU="40" Memory="100" />
    </paaS>
    <paaS id="2">
      <platform xsi:type="SCloud:Apusic" VID="3" Storage="44" CPU="22" Memory="50" />
      <platform xsi:type="SCloud:Tomcat" VID="4" Storage="120" CPU="60" Memory="90" ThreadPool="5"/>
    </paaS>
  </paaSList>
  <nodeList>
    <node ID="1">
      <VM xsi:type="SCloud:XenVM" VID="1"/>
      <VM xsi:type="SCloud:XenVM" VID="3"/>
    </node>
    <node ID="2">
      <VM xsi:type="SCloud:XenVM" VID="2"/>
      <VM xsi:type="SCloud:XenVM" VID="4"/>
    </node>
  </nodeList>
</SCloud:Cloud>
```

系统片段定制描述

```
<Cloud>
  <PaasList>
    <Paas id="2">
      <Platform vid="3">
        <property>CPU</property>
        <property>Memory</property>
        <property>Storage</property>
      </Platform>
      <Platform vid="4">
        <property>Memory</property>
        <property>ThreadPool</property>
      </Platform>
    </Paas>
  </PaasList>
</Cloud>
```

Fig.4 Fragments recognition in cloud resource runtime models

图 4 云资源运行时模型中片段的识别

组合模型中的模型片段来自不同的云资源运行时模型,它们不存在语法上的关联.因此,我们将每一个模型片段的根节点作为组合模型根节点的直接子节点.同时,为了解决各个模型片段的命名冲突,我们还需要维护一个命名空间,对冲突的命名进行替换和记录.在组合模型内部,元素采用新的不会冲突的命名,而当组合模型与运行时模型发生交互时,被更改命名的元素则依据命名空间恢复原来的命名,使得能够被运行时模型识别,如图 5 所示.

组合模型

```
<merged:Merged>
  <projects mark="162.105.31.245_GCCloud">
    <project project_id="1" creationTimestamp="28902342133">
      <images project_id="1">
        <image id="1" creationTimestamp="79234123" name="image_1" />
        ...
      </images>
      ...
    </project>
  </projects>
  <op_projects mark="222.20.11.4_OPCloud">
    <op_project tenant_id="1">
      <op_images>
        <op_image id="1" minDisk="512MB" minRam="512MB"/>
        ...
      </op_images>
      ...
    </op_project>
  </op_projects>
</merged:Merged>
```

模型片段

```
<OPCloud:Projects ip="222.20.11.4">
  <project tenant_id="1">
    <images>
      <image id="1" minDisk="512MB" minRam="512MB"/>
      ...
    </images>
    ...
  </project>
</OPCloud:Projects>
```

命名空间

```
<namespace_description>
  <google_compute>
    <projects />
    <images />
    <image />
  </google_compute>
  <open_stack>
    <projects alias="op_projects" />
    <images alias="op_images" />
    <image alias="op_image" />
  </open_stack>
</namespace_description>
```

Fig.5 Solution to naming conflict in composite models

图 5 组合模型中元素命名冲突的解决方法

3.2 数据同步

在模型定制的基础上,只有维护组合模型与系统运行时模型的数据同步,才能实现组合模型对多个系统的统一管理.组合模型由不同的系统运行时模型元素构成,其数据一致性由不同的模型片段分别完成.模型片段通过模型操作的传输和执行来实现其数据的同步.一方面,当管理员对组合模型进行操作时,其模型操作会发送给系统运行时模型并执行;另一方面,在系统运行时模型处部署一个模型片段副本,通过轮询机制比较发现模型的变化,自动生成相应的模型操作,发送给组合模型并执行.

当管理员对组合模型进行写操作时,为了维护组合模型与系统运行时模型的数据的一致性,其模型操作会发送给系统运行时模型并执行.在所有模型操作类型中,只有 Set,Add 和 Remove 操作能够产生相应的模型变化,图 6 对其描述规则和产生效果进行了明确的定义.

Add	Description	⟨action node="TypeS", type="add"⟩ ⟨query node="TypeF", condition="Constraint"/⟩ ⟨set key="·", value="·"/⟩ ⟨set key="·", value="·"/⟩ ⟨/action⟩
	Post condition	∃TypeS *s*, ∃TypeF *f*, *s*∈*f*∧*f* inconditionof Constraint∧props⊆*s.properties*
	Illustration	在模型中,寻找类型为"TypeF"且满足约束"Constraint"的元素,以其为父节点, 添加一个类型为"TypeS"的子节点,并为其属性赋值
Set	Description	⟨action key="·", value="·", type="set"⟩ ⟨query node="Type", condition="Constraint"/⟩ ⟨/action⟩
	Post condition	∃Type *n*, *n* inconditionof Constraint∧prop∈*n.properties*
	Illustration	在模型中,寻找类型为"Type"且满足约束"Constraint"的元素,将其属性 key 的值赋为 value
Remove	Description	⟨action node="Type", condition="Constraint", type="remove"/⟩
	Post condition	∀Type *n*, *n* notinconditionof Constraint
	Illustration	在模型中,寻找类型为"Type"且满足约束"Constraint"的元素,将其删除

Fig.6 Three kinds of model operations

图 6 3 种模型操作

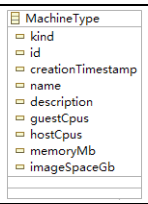
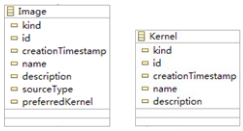
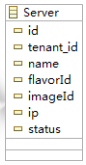
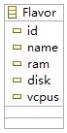
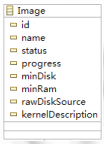
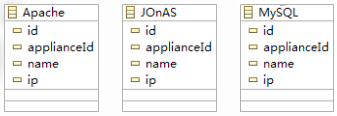
当系统运行时模型发生变化时,为了维护组合模型与系统运行时模型的数据的一致性,需要能够感知这一变化,并将其反馈给组合模型.为了实现这一功能,我们在系统运行时模型处部署一个模型片段副本,通过轮询机制比较发现模型的变化,以自动生成相应的模型操作,发送给组合模型并执行.这一过程的难点在于模型变化的发现和相应模型操作的生成,我们是通过深度优先算法来实现的:

a) 根据模型片段的定制描述,在系统运行时模型中抽取新的模型片段;
b) 从旧模型片段的根节点开始,依次对每一个节点在新旧模型片段中进行比较:若该节点是根节点,则比较是否相同,若不同,报错,若相同,则执行步骤 d);若该节点不是根节点,则执行步骤 c);
c) 检查新的模型片段中是否有此节点,若有,执行步骤 d);若没有,则根据该子节点信息,生成相应的"Remove"操作,执行步骤 f);
d) 检查新的模型片段中此节点的属性值是否有变化,若没有,执行步骤 e);若有,则根据该子节点信息,生成相应的"Set"操作,执行步骤 e);
e) 在新旧模型片段中,对此节点的子节点进行比较,检查新的模型片段中是否有子节点的增加:若没有,执行步骤 f);若有,则根据该子节点信息,生成相应的"Add"操作,执行步骤 f);
f) 在旧模型片段中寻找下一个节点,若存在下一个节点,重复步骤 b);若不存在下一个节点,则算法结束.

4 组合模型到用户特定模型的转换方法

云资源的管理需要结合管理员的知识、经验和特定的管理场景,以满足不同的用户偏好.图 1 中,用户特定模型表示特定的用户偏好,而组合模型则表示受管资源的集合,需要建立它们之间的关联,使得管理员能够通过用户特定模型对云资源进行管理.本文方法中,我们通过建立用户特定模型与组合模型间的元素映射关系,来实

现模型的双向转换.任何一个用户特定模型中的元素属性与一个组合模型中相对应的元素属性保持值的相等,且任何一个用户特定模型上的模型操作转换为一个组合模型上相应的模型操作,以达到预期的管理效果.特别地,用户特定模型与组合模型间的元素映射关系需要编写模型转换程序来实现.我们完成了一种模型转换方法,能够根据用户特定模型、组合模型及它们之间的元素映射关系,自动生成相应的模型转换程序.图 7 展示了模型元素间的 3 种基本映射关系.

	“一对一”映射关系	“多对一”映射关系	“一对多”映射关系
源模型中的类			
目标模型中的类			

转换代码示例	vcpus:=self.guestCpus	kernelDescription:=
		gccloud.objectsOfType(Kernel)→
		select(id=self.preferredKernel)→
		selectOne(true).description
			if (self.imageId=“MYSQLTYPE”) {
			return object MySQL {…}
			}

Fig.7 Three types of basic mapping rules between model elements

图 7 模型元素间的 3 种基本映射关系

(1) 模型元素间“一对一”映射关系

源模型中的一个类与目标模型中的一个类对应,特别地,目标模型中类的属性可以在源模型中对应的类中找到对应的属性.它们通常是指源模型中存在一种类型的元素,目标模型中也存在一种类型的元素,它们均是为了描述同一类型的事物.在源模型向目标模型发生转换时,目标模型中元素的属性会根据其在源模型中关联元素的对应属性进行赋值.例如,源模型中的 MachineType 类与目标模型中的 Flavor 类均表示虚拟机计算资源的配置情况,且 Flavor 类的属性 id,name,ram,disk 和 vcpus 在 MachineType 类中存在对应的属性 id,name,memoryMb,imageSpaceGb 和 guestCpus.因此,MachineType 类与 Flavor 类存在“一对一”的映射关系.

(2) 模型元素间“多对一”映射关系

源模型中的两个或多个类与目标模型中的一个类对应,特别地,目标模型中类的属性在源模型中的对应属性分布在两个或多个类中.它们通常是指源模型中存在两种或两种以上的元素,共同描述某一事物,而目标模型中仅用一种元素来描述这一事物.在源模型向目标模型发生转换时,目标模型中的元素需要在源模型中查询与之相对应的两个或多个元素来进行其属性的赋值.例如,源模型中的 Image 类与目标模型中的 Image 类均表示虚拟机类型,但是目标模型的 Image 类中的 kernelDescription 属性无法在源模型的 Image 类中找到对应属性,其对应属性在目标模型中的 Kernel 类中(模型转换时,可以在源模型中查找得到 id 为 Image 元素 preferredKernel 属性值的 Kernel 元素).

(3) 模型元素间的“一对多”映射关系

源模型中的一个类与目标模型中的两个或多个类对应.它们通常是指源模型中的一种类型的元素用来描述某一类的事务,而目标模型中的两种或多种类型的元素分别描述该类事务的不同子类型事务.当模型转换时,源模型中的元素需要根据自身的属性信息选择目标模型中特定一种类型的元素进行映射.例如,源模型中的 Server 类表示虚拟机,目标模型中的 Apache 类、JOnAS 类和 MySQL 类表示预装相应软件的虚拟机.模型转换时,源模型中任何一个 Server 元素会根据其属性 imageId 的值(反映其虚拟机类型)映射为目标模型中 Apache,JOnAS 或 MySQL 类型的一个元素.

用户特定模型与组合模型间的元素映射关系均可以表示为以上 3 种基本映射关系的组合.

我们设计了一套映射关系描述规则及其自动生成代码的方法,使得管理员能够通过定义模型间的元素映射关系来得到相应的模型转换程序(QVT 语言[8]).模型间的元素映射关系通过一个 XML 文件进行描述,描述规则中的关键字定义如下:

(1) helper:用于描述类和类之间元素的映射关系

helper 标签一般含有两个属性:一个是 key,一个是 value.value 表示源模型中将要映射的元素,key 表示目标模型中对应的元素.

helper 通常还有一个属性 type 来表明从源模型到目标模型的元素映射类型,存在如下两种 type 取值:

- 当 type 为“basic”时,表示该 helper 代表的是元素间“一对一”映射关系或“多对一”映射关系;
- 当 type 为“multi”时,表示该 helper 代表的是元素间的“一对多”映射关系,此时,helper 往往成组出现.

由于 helper 用于描述元素的映射关系,而元素通常还会包含属性或其他元素,因此,helper 标签里通常会嵌套 helper 标签、mapper 标签和 query 标签,其中,mapper 标签和 query 标签均表示属性间的映射关系.

(2) mapper:用于描述类和类之间属性的映射关系

mapper 标签通常含有两个属性:key 和 value.key 表示目标模型应该被映射的元素属性的名称,而 value 表示源模型中元素对应属性的名称,其中,它们所归属的元素分别由上一层的 helper 标签定义.

(3) query:用于描述类和类之间属性的映射关系

与 mapper 类似,query 标签也有 key 和 value 属性,其中,key 表示目标模型应该被映射的元素属性的名称,而 value 表示源模型中元素对应属性的名称,其中,

- 目标模型中属性所归属的元素类型由上一层的 helper 标签定义;
- 而源模型中属性所归属的元素则由 query 标签的其他两个属性 node 和 condition 定义,它们分别表示元素的类型和需要满足的条件.

如图 8 所示,在以上关键字的基础上,我们就可以按照“一对一”映射关系、“多对一”映射关系和“一对多”映射关系的定义,分别对模型间的元素映射关系进行描述.

1. 在“一对一”映射的例子中,源模型中 Flavor 类的元素映射到目标模型中 MachineType 类的元素,因此用一个 helper 标签来描述,helper 标签的 key 属性和 value 属性分别为目标模型和源模型的类的名称.
   - Flavor 的 id 与 MachineType 的 id 对应;
   - name 和 name 对应;
   - ram 和 memoryMb 对应;
   - disk 和 imageSpaceGb 对应;
   - vcpus 和 guestCpus 对应.

   因此用 mapper 标签来描述,mapper 标签的 key 属性和 value 属性分别为目标模型和源模型中对应属性的名称;
2. 在“多对一”映射的例子中,源模型中的 Image 类和 Kernel 类的元素映射到目标模型中的 Image 类的元素,源模型中的 Image 类和目标模型的 Image 类的相似度比较大,因此我们用一个 helper 标签来描述这两个类的元素间的映射关系.源模型中另一个 Kernel 类的元素属性到目标模型 Image 类的元素属性的映射,我们用 query 标签来描述,query 标签的 key 属性和 value 属性分别表示目标模型和源模型中对应属性的名称,其中,node 属性值为 Kernel,表示源模型中属性所归属元素的类型为 Kernel,condition 属性值则表示元素需要满足的条件;
3. 在“一对多”映射的例子中,源模型中的 Server 类的元素可能映射为目标模型中的 Apache 类、JOnAS 类、MySQL 类的元素中的一种.我们用 type 值为“multi”的 helper 标签来描述“一对多”的映射关系,其中,condition 属性值为“self.imageId=1”,表明 Server 元素的 imageId 为 1 是该映射的发生条件.

Fig.8 Descriptions of basic mapping rules between model elements

图 8 模型元素间基本映射关系的描述

5 实例研究

云环境中,需要对虚拟机硬件资源及其上基础软件同时进行管理,以实现资源优化配置.目前,一些开源软件针对云计算中的特定管理问题提出了自己的解决方案,其中,OpenStack 是一款开源的基础设施管理软件,提供了对虚拟机硬件资源进行配置的管理接口;Hyperic 则是一款开源的中间件管理软件,提供了对多种中间件软件进行配置的管理接口.然而,目前不存在一套成熟的同时对虚拟机软、硬件资源进行管理的开源方案.

为了验证本文方法的可行性和有效性,我们基于 OpenStack 与 Hyperic 两款独立的管理软件,通过模型构造、合并和转换的方法,实现了虚拟机软、硬件资源的统一管理.首先,构造了 OpenStack 与 Hyperic 的运行时模型;其次,通过组合模型对两个运行时模型进行了合并;最后,根据管理需求构造了用户特定模型,并实现了组合模型和用户特定模型的双向转换.此外,我们还将整体解决方案与传统管理模式进行了比较,对方法的可行性和有效性进行了评估.

5.1 OpenStack与Hyperic运行时模型

OpenStack 能够对虚拟机的整个生命周期进行管理,其资源分配的最小单元是虚拟机(server),而每个虚拟机都存在于一个项目(project)中,整个基础设施的资源按照项目分成多个部分;每一个虚拟机包含一个映像文

件(image)和一个配置文件(flavor),它们分别对虚拟机的文件系统和硬件资源进行配置.图 9 描述了 OpenStack 包含的主要元素:Projects 是根元素,描述了控制节点的基本信息,并包含了项目的列表,Project 则表示一个项目,包含一个 Images 元素、一个 Flavors 元素和一个 Servers 元素;Images 元素包含了 Image 的列表,表示该项目可以使用的虚拟机映像文件的集合,而每一个 Image 元素都表示了一种虚拟机映像(文件系统);Flavors 元素包含了 Flavor 的列表,表示该项目可以使用的虚拟机配置文件的集合,而每一个 Flavor 元素都表示了一种虚拟机配置(CPU,Memory 等资源).OpenStack 的操作主要包括以上各种元素的添加、删除和属性查看、修改以及元素状态的改变,这些操作分别可以映射为元素的增删、属性的查改以及属性 status 的变化.例如,虚拟机的创建、删除映射为 Server 元素的增删,虚拟机名字的查看、更改映射为 Server 中 name 属性的查改,而虚拟机的暂停、重启则映射为 Server 中 status 属性的变化.

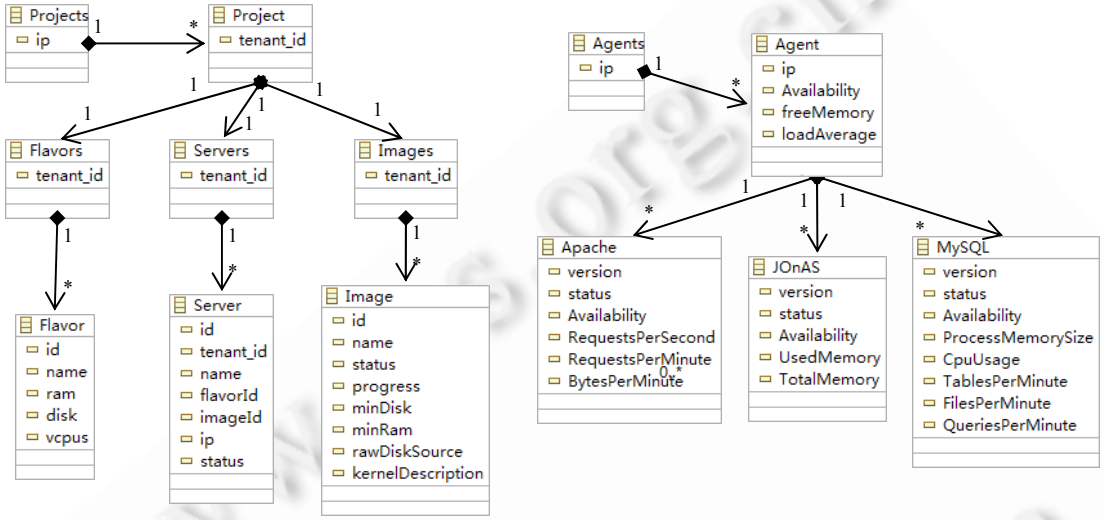


Fig.9 Architecture-Based models of OpenStack (Left) and Hyperic (Right)

图 9 OpenStack 体系结构模型(左)与 Hyperic 体系结构模型(右)

Hyperic 在每个受管节点上安装一个代理控制器(agent),通过代理控制器能够对节点上的多种中间件进行管理.由于中间件数量较多,其属性数量巨大,实例中仅对 Apache,JOnAS 和 MySQL 这 3 种中间件及它们的默认监测属性进行了管理接口的封装.图 9 描述了 Hyperic 包含的主要元素:Agents 是根元素,描述了控制节点的基本信息,并包含了代理控制器的列表,Agent 则表示一个代理控制器,包含一个 Apache 元素、一个 JOnAS 元素和一个 MySQL 元素;Apache 元素、JOnAS 元素和 MySQL 元素分别表示了节点上可能安装的 3 种中间件,它们的属性则表示了中间件的性能指标和配置参数.Hyperic 的操作主要是中间件性能指标和配置参数的读写,它们则映射为元素属性的查改.

基于 OpenStack 与 Hyperic 运行时模型,我们可以在模型层通过模型操作,分别对虚拟机硬件资源及其上基础软件进行管理.

5.2 OpenStack与Hyperic组合模型

OpenStack 模型能够对虚拟机硬件资源进行管理,而 Hyperic 模型则能够对虚拟机上基础软件进行管理.将以上两个模型作为模型片段,组合成为一个新的模型,组合模型就能够对虚拟机软、硬件资源进行统一的管理.通过管理操作的传输和执行,即可以实现新模型与 OpenStack 运行时模型、Hyperic 运行时模型的数据同步.

图 10 展示了虚拟机创建操作传输到 OpenStack 运行时模型并被执行的过程.在虚拟机创建操作执行之前,OpenStack 系统中有两个项目,其中一个项目包含一个虚拟机,而另一个项目则不包含虚拟机.传输到运行时模型的操作文件对将要执行的操作进行了详细的描述.

(1) Query:找到一个项目,其 tenant_id 为 f9764071;

(2) Add:创建一个元素,其类型(node)为虚拟机(sever);

(3) Set:新建元素的 name 属性(表示虚拟机的名字)的值为 JOnASAna,flavorId 属性(表示虚拟机的配置文件)的值为 2,imageId 属性(表示虚拟机的映像)的值为 6ebf952c.

根据操作文件的描述,运行时模型进行了相应的虚拟机创建操作;而虚拟机的固有属性(id,ip 等)则在执行创建操作后,由 OpenStack 系统自动给出.

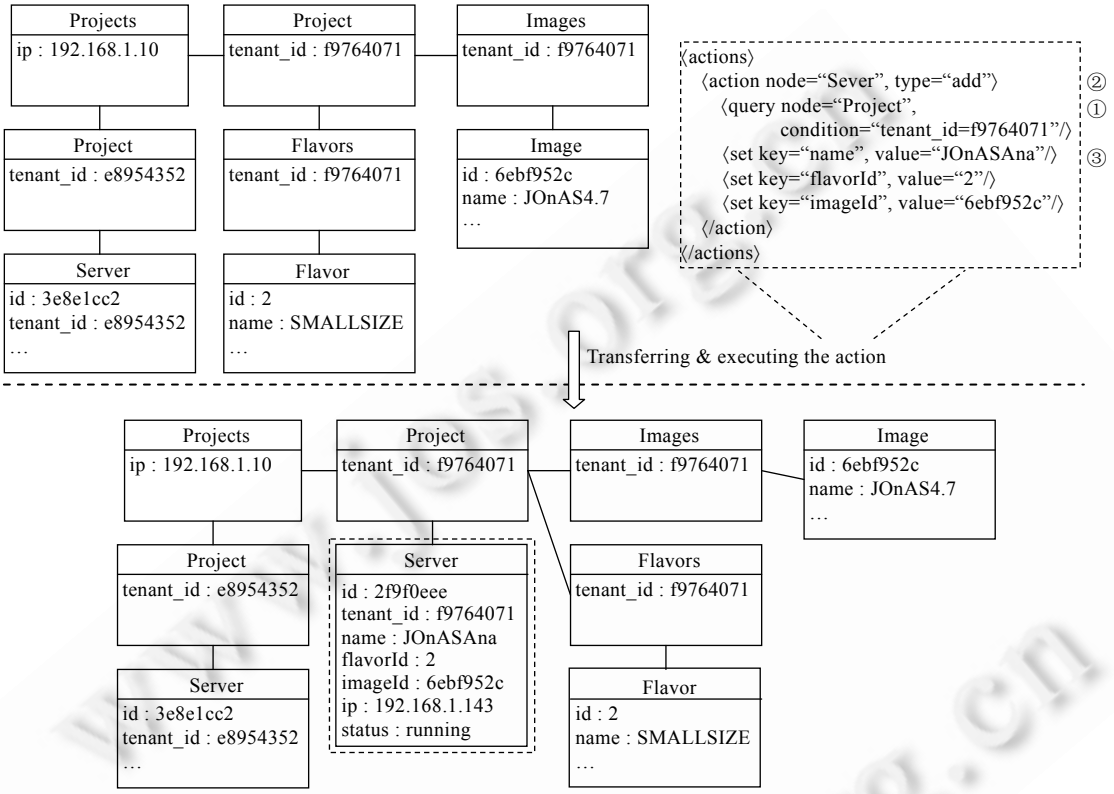


Fig.10 Operation of virtual machine creation being executed on the runtime model of OpenStack

图 10 虚拟机创建操作在 OpenStack 运行时模型上的执行

5.3 OpenStack与Hyperic需求模型到组合模型的映射

云环境中,需要对虚拟机硬件资源及其上基础软件同时进行管理,以实现资源优化配置.管理员把基础软件及其所在的虚拟机看作一个"设备"(virtual application[9]),作为最小的受管单元,而每一个设备存在于一个项目中,以共同服务于特定的应用系统,整个基础设施的资源则按照项目分成多个部分.针对以上场景,我们以 Apache,JOnAS 和 MySQL 这 3 类"设备"为例,构造了用户特定模型.图 11 描述了用户特定模型包含的主要元素:Projects 是根元素,并包含了项目列表,Project 元素则表示一个项目,包含一个 Servers 元素;Servers 元素表示项目拥有的设备总和,包含一个 Apache 列表、一个 JOnAS 列表和一个 MySQL 列表;Apache 元素表示 Apache 类型的设备,包含一个 ApacheSwConfig 元素,表示其软件参数配置(以常见属性为例),还包含一个 HwConfig 元素,表示其硬件资源配置;JOnAS 和 MySQL 则与 Apache 类似.因此,软、硬件资源统一管理过程中的设备增删和参数查改等管理操作均可以通过用户特定模型中元素增删及属性查改等模型操作来表示.

用户特定模型对管理需求进行了清晰的定义,而 OpenStack 和 Hyperic 的组合模型则反映了目标受管资源的集合.其中,用户特定模型中 Apache 元素、JOnAS 元素和 MySQL 元素(3 种类型的设备)与组合模型中 Server

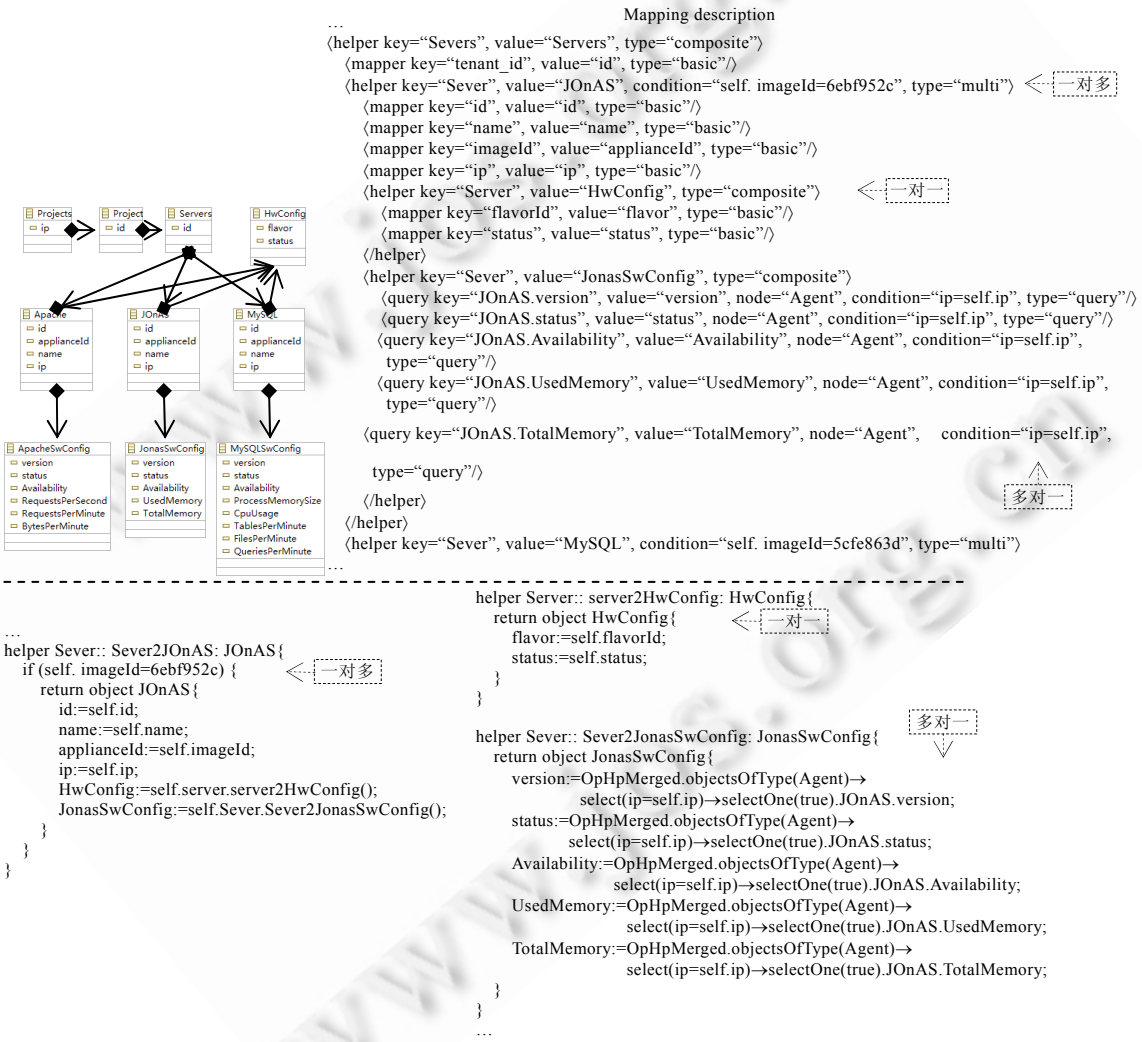
元素(虚拟机)、Agent 元素之间的映射关系是整个模型转换的关键.下面以组合模型(作为源模型)中 Server 元素和 Agent 元素到用户特定模型(作为目标模型)中 JOnAS 元素的映射为例子,来详细介绍映射关系的描述及代码的生成.

组合模型中(如图 9 所示),Server 元素表示虚拟机,它包含虚拟机映像和资源配置等信息,Agent 元素表示代理控制器,能够对其节点上的 JOnAS 进行管理;而用户特定模型中,JOnAS 元素表示设备,包含软、硬件配置信息.因此,该映射实际上是 Server 元素中硬件资源信息和 Agent 元素中软件参数信息到 JOnAS 元素的映射.此过程中,主要存在两个难点:

(1) Server 元素可能映射为 Apache 元素、JOnAS 元素和 MySQL 元素中的一种,需要根据虚拟机类型(condition="self.imageId=TYPE")来确定对应元素的类型,即"一对多"的映射关系;

(2) 根据 Server 元素映射得到的 JOnAS 元素,还需要从相对应的 Agent 元素(condition="ip=self.ip")获取软件参数信息,需要根据虚拟机位置来确定关联的 Agent 元素,即"多对一"的映射关系.

图 11 展示了 Server 元素和 Agent 元素到 JOnAS 元素的映射关系描述及代码片段.




```
…
⟨helper key="Severs", value="Servers", type="composite"⟩
  ⟨mapper key="tenant_id", value="id", type="basic"/⟩
  ⟨helper key="Sever", value="JOnAS", condition="self. imageId=6ebf952c", type="multi"⟩
    ⟨mapper key="id", value="id", type="basic"/⟩
    ⟨mapper key="name", value="name", type="basic"/⟩
    ⟨mapper key="imageId", value="applianceId", type="basic"/⟩
    ⟨mapper key="ip", value="ip", type="basic"/⟩
    ⟨helper key="Server", value="HwConfig", type="composite"⟩
      ⟨mapper key="flavorId", value="flavor", type="basic"/⟩
      ⟨mapper key="status", value="status", type="basic"/⟩
    ⟨/helper⟩
    ⟨helper key="Sever", value="JonasSwConfig", type="composite"⟩
      ⟨query key="JOnAS.version", value="version", node="Agent", condition="ip=self.ip", type="query"/⟩
      ⟨query key="JOnAS.status", value="status", node="Agent", condition="ip=self.ip", type="query"/⟩
      ⟨query key="JOnAS.Availability", value="Availability", node="Agent", condition="ip=self.ip",
        type="query"/⟩
      ⟨query key="JOnAS.UsedMemory", value="UsedMemory", node="Agent", condition="ip=self.ip",
        type="query"/⟩
      ⟨query key="JOnAS.TotalMemory", value="TotalMemory", node="Agent",  condition="ip=self.ip",
        type="query"/⟩
    ⟨/helper⟩
  ⟨/helper⟩
  ⟨helper key="Sever", value="MySQL", condition="self. imageId=5cfe863d", type="multi"⟩
…
```

```
…
helper Sever:: Sever2JOnAS: JOnAS{
  if (self. imageId=6ebf952c) {
    return object JOnAS{
      id:=self.id;
      name:=self.name;
      applianceId:=self.imageId;
      ip:=self.ip;
      HwConfig:=self.server.server2HwConfig();
      JonasSwConfig:=self.Sever.Sever2JonasSwConfig();
    }
  }
}
```

```
helper Server:: server2HwConfig: HwConfig{
  return object HwConfig{
    flavor:=self.flavorId;
    status:=self.status;
  }
}

helper Sever:: Sever2JonasSwConfig: JonasSwConfig{
  return object JonasSwConfig{
    version:=OpHpMerged.objectsOfType(Agent)→
        select(ip=self.ip)→selectOne(true).JOnAS.version;
    status:=OpHpMerged.objectsOfType(Agent)→
        select(ip=self.ip)→selectOne(true).JOnAS.status;
    Availability:=OpHpMerged.objectsOfType(Agent)→
        select(ip=self.ip)→selectOne(true).JOnAS.Availability;
    UsedMemory:=OpHpMerged.objectsOfType(Agent)→
        select(ip=self.ip)→selectOne(true).JOnAS.UsedMemory;
    TotalMemory:=OpHpMerged.objectsOfType(Agent)→
        select(ip=self.ip)→selectOne(true).JOnAS.TotalMemory;
  }
}
…
```

Fig.11 Customized model, mapping descriptions and code snippets

图 11 用户特定模型、映射关系描述及代码片段

通过用户特定模型与组合模型间的双向转换,用户特定模型上的任何操作可以自动映射为组合模型上的模型变化.同时,组合模型上的任何变化也能够自动反映在用户特定模型上.因此,管理员能够通过用户特定模型对受管资源进行管理.

5.4 效果评估

我们开发了基于模型的虚拟机软、硬件资源统一管理工具.如图 12 所示,管理员通过用户特定模型进行云资源的管理.每个设备元素代表一个运行着特定类型基础软件的虚拟机,这些设备元素共同构成运行系统模型.通过 OpenStack 和 Hyperic 的管理接口,运行系统模型将转化为真实环境中运行着不同类型基础软件的多个虚拟机,而这些虚拟机的状态及其硬件资源和软件参数配置,则与运行系统模型中的设备元素保持一致.基于我们的工具,管理员能够通过运行系统模型对实际系统进行管理.特别地,工具没有对 OpenStack 和 Hyperic 进行任何修改,仅仅是在它们的运行时模型基础上对其管理接口进行了复用和重组织.

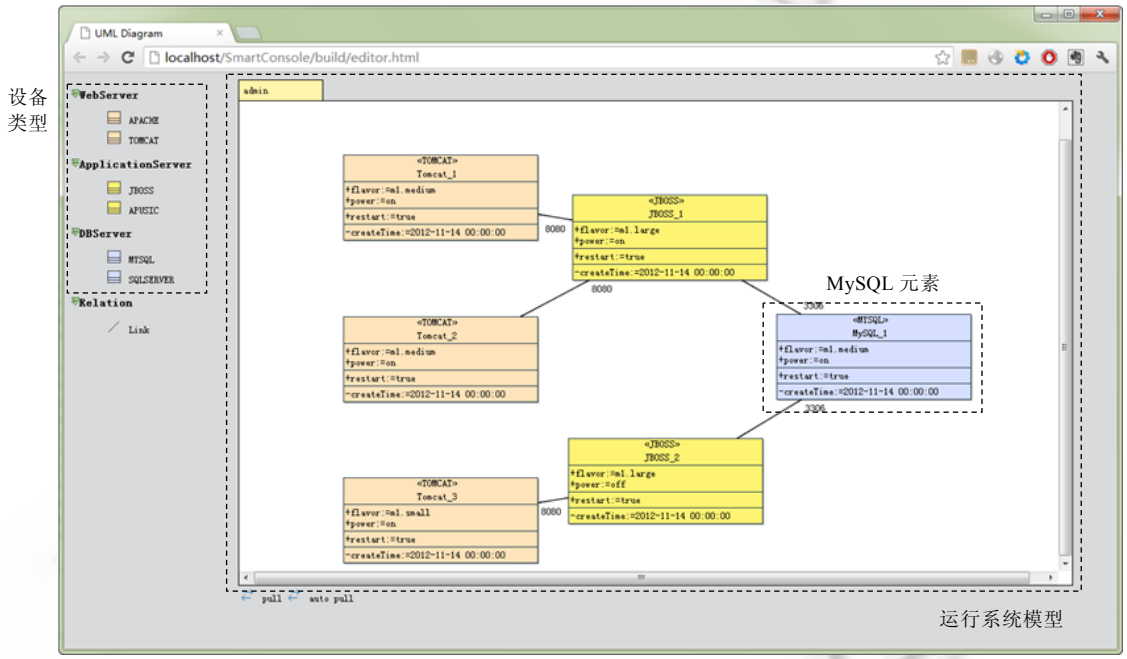


Fig.12 Runtime model-based tool of unified management of virtual machine resources

图 12 基于运行时模型的虚拟机软硬件资源统一管理工具

进一步地,我们使用统一管理工具,对一个具有 11 台物理服务器并支持 100 个虚拟设备运行的云进行管理,还构造了两组管理任务,对其系统读取和写入能力进行性能测试.基于底层管理接口和统一管理工具,我们分别编写了相应的 Java 程序和 QVT 程序,并对它们的执行结果进行了比较,如图 13 所示.其中,执行时间是指完成管理任务所耗费的平均时间,而数据延迟是指获得数据的平均延迟时间(例如,数据延迟为 30s,是指系统 30s 前的数据).

第 1 组管理任务是查询虚拟设备的特定属性,如图 13 所示,我们得到了目标设备数目分别为 5 个、20 个和 100 个时的读取任务执行结果,其中,Java 程序的平均执行时间较长,而 QVT 程序的数据延迟较大.一方面,Java 程序通过调用管理接口来逐个读取每个设备的属性信息,其复杂度和延迟时间均与管理接口的调用次数呈线性关系;另一方面,本文方法中的模型片段相当于系统实时数据的快照,QVT 程序直接读取片段中记录的设备属性信息,执行时间较短.但是,模型片段会产生固定的数据延迟,相当于从运行系统读取一次模型中所有数据花费的时间,与模型的规模成正比.然而,从系统管理的角度来看,这种由模型同步导致的数据延迟是可以接受的.

管理任务	目标设备数目(单位:个)	底层管理接口		统一管理工具	
		执行时间(单位:秒)	数据延迟(单位:秒)	执行时间(单位:秒)	数据延迟(单位:秒)
设备数据查询(系统读取任务)	5	1.2	0.6	0.6	30
	20	4.2	2.1	0.8	30
	100	20	10	1.6	30
创建虚拟设备(系统写入任务)	1	0.2	–	0.5	–
	5	1.0	–	1.8	–
	20	4.0	–	6.2	–

Fig.13 Performance test results of the unified management tool

图 13 统一管理工具的性能测试结果

第 2 组管理任务是创建一定数量的虚拟设备,如图 13 所示,我们得到了目标设备数目分别为 1 个、5 个和 20 个时的写入任务执行结果,其中,QVT 程序的执行时间略长于 Java 程序.统一管理工具的管理功能基于底层管理接口,并且还需要一些额外的操作来支持模型与运行系统的同步.因此,执行相同的操作需要花费更多的时间.然而从系统管理的角度来看,它们性能上的差别并不是很大,甚至是可以被忽略的.

基于本文方法,我们在 OpenStack 和 Hyperic 两种管理软件的基础上构造了虚拟机软、硬件资源的统一管理工具,来验证方法的可行性和有效性.

- 一方面,在此过程中,我们通过模型的构造、合并和转换,对已有管理软件进行了集成,并满足了目标管理需求,由此验证了基于运行时模型的方法,使得开发人员能够在体系结构的层次上进行新的管理工具的开发,而不需要进行系统管理功能调用和数据交互等底层代码的开发,能够极大地降低开发难度和复杂度;
- 另一方面,通过与传统方法的比较,我们验证了本文方法给管理任务的执行所带来的影响是可控的.特别地,从系统管理的角度来看,这些影响甚至是可以被忽略的.

6 相关工作

尽管学术界和产业界都对云管理进行了大量的研究,但资源的多样性和服务的随需性,使得单个管理方案无法满足不断涌现的管理需求,导致存在大量的云管理系统,例如 Eucalyptus[10],OpenStack[11],Tivoli[12]和 Hyperic[13]等,它们往往针对同一类型的资源或是特定的管理场景,却无法有效地根据受管资源和用户偏好的变化而进行调整.

目前存在一些研究工作,试图基于面向服务的体系结构来解决管理功能的集成问题.文献[14]提出一种在分布式环境下对不同系统进行管理的方法,其将系统管理功能以 RESTful 服务的形式发布出来,并使用订阅机制实现分布式管理功能的集成,最终实现不同系统间的统一管理.在文献[15]中,我们则基于复用的观点提出“管理功能服务化”,将 IT 管理功能、管理流程和管理经验封装为 SOAP 服务,并对它们进行组装和复用,以服务流程的形式达到同时对多个系统进行管理的目的.尽管面向服务的体系结构能够解决系统间的统一管理问题,管理功能服务却不如系统参数那样能够直观地反映系统运行状态,而服务的订阅和组装也比较复杂,这些均会给系统管理带来额外的困难.

在以往的研究中,基于体系结构模型的方法常常被用于系统的管理.例如在文献[16]中,它被用于网络设备配置正确性的自动诊断,该工作对受管元素的相关特征进行建模并对它们间的约束关系进行定义,使得能够自动地对受管元素当前配置的正确性进行计算.文献[17]则面向特定类型的系统,对其基础设施建立统一模型,通过收集系统运行时状态、系统各模块间的依赖以及系统目标等信息,自动地对系统运行环境的资源配置进行计算.虽然基于体系结构模型的方法能够为特定场景的管理提供良好的解决方案,但是以往的研究工作还存在以下两方面的不足:

(1) 体系结构模型与运行系统相分离,体系结构模型只能用于系统管理问题的计算,而不能用于具体管

理操作的实施;

(2) 对于每一个管理场景,需要建立全新的体系结构模型,开发代价较高.

在之前的工作中,作者所在研究团队提出了 SM@RT 工具[5],支持运行时模型的自动生成以及与系统状态的自动同步,使得能够通过对模型进行操作来管理目标系统;进一步地,我们还分别构建了 JOnAS、云基础设施等运行时体系结构模型[6,7],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治管理.在这些工作的基础上,本文提出了基于运行时模型的多样化云资源管理方法,能够根据管理需求快速定制、集成、扩展已有的管理软件,降低了开发难度和复杂度.

7 结束语

云管理中,多样化的受管资源和不断变化的管理需求使得单个管理方案无法满足不断涌现的管理需求,导致存在大量的云管理系统.针对一个特定的管理需求,目前主要采用的管理方式是基于已有管理软件进行二次开发得到一个新的能够满足管理需求的管理系统.为了能够根据管理需求快速定制、集成、扩展已有的管理软件,本文将运行时模型引入到云管理过程中,建立了基于运行时模型的多样化云资源管理方法.该方法通过构造运行时模型实现在模型层对单一受管资源进行管理,通过模型合并实现多种受管资源的统一管理,通过模型转换实现用偏好的方式对云资源进行管理.

未来工作的重点主要包含两个方面:

- 一方面,将方法运用到实际生产环境中;
- 另一方面,在本文方法的基础上进行管理策略的研究,基于模型分析、推理等技术,实现系统容错、容量规划等高级管理功能,从而进一步简化云管理.

References:

[1] Mell P, Grance T. The NIST definition of cloud computing. Technical Report, Special Publication 800-145, U.S. Department of Commerce, National Institute of Standards and Technology, 2009.

[2] Garlan D. Software architecture: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 91−101.

[3] Mei H, Shen JR. Progress of research on software architecture. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[4] France R, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Proc. of the 29th Int'l Conf. on Software Engineering, Future of Software Engineering Track. Washington: IEEE Computer Society Press, 2007. 37−54. [doi: 10.1109/FOSE.2007.14]

[5] Huang G, Song H, Mei H. SM@RT: Applying architecture-based runtime management of internetware systems. Int'l Journal of Software and Informatics, 2009,3(4):439−464. [doi: 10.1145/1640206.1640215]

[6] Song H, Huang G, Chauvel F, Xiong YF, Hu ZJ, Sun YC, Mei H. Supporting runtime software architecture: A bidirectional-transformation-based approach. Journal of Systems and Software, 2011,84(5):711−723. [doi: 10.1016/j.jss.2010.12.009]

[7] Huang G, Chen X, Zhang Y, Zhang XD. Towards architecture-based management of platforms in the cloud. Frontiers of Computer Science, 2012,6(4):388−397. [doi: 10.1007/s11704-012-2100-4]

[8] Object Management Group. Meta object facility (MOF) 2.0 query/view/transformation (QVT). http://www.omg.org/spec/QVT

[9] Wikipedia. Virtual appliance. http://en.wikipedia.org/wiki/Virtual_appliance

[10] Nurmi D. The eucalyptus open-source cloud-computing system. In: Proc. of the 9th IEEE/ACM Int'l Symp. on Cluster Computing and the Grid. 2009. 124−131. [doi: 10.1109/CCGRID.2009.93]

[11] OpenStack. The open source cloud operating system. 2013. http://openstack.org/projects/

[12] IBM. IBM Tivoli software. 2013. http://www-01.ibm.com/software/tivoli/

[13] SpringSource. Hyperic. 2013. http://www.hyperic.com/

[14] Ludwig H, Laredo J, Bhattacharya K. Rest-Based management of loosely coupled services. In: Proc. of the 18th Int'l Conf. on World Wide Web. New York: ACM Press, 2009. 931−940. [doi: 10.1145/1526709.1526834]

[15] Chen X, Liu XZ, Fang FZ, Zhang XD, Huang G. Management as a service: An empirical case study in the internetware cloud. In: Proc. of the 7th IEEE Int'l Conf. on E-Business Engineering. 2010. 470−473. [doi: 10.1109/ICEBE.2010.53]

[16] Hallé S, Wenaas E, Villemaire R, Cherkaoui O. Self-Configuration of network devices with configuration logic. In: Proc. of the 1st IFIP TC6 Int'l Conf. on Autonomic Networking. Paris: Springer-Verlag, 2006. 36−49. http://link.springer.com/chapter/10.1007/11880905_4

[17] Cuadrado F, Dueñas JC, García-Carmona R. An autonomous engine for services configuration and deployment. IEEE Trans. on Software Engineering, 2012,38(3):520−536. [doi: 10.1109/TSE.2011.24]

附中文参考文献:

[3] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

陈星(1985－),男,福建永春人,博士,主要研究领域为分布式系统,软件中间件,软件工程.
E-mail: pkubluestar@gmail.com

武义涵(1988－),男,学士,主要研究领域为运行时模型,软件可靠性与高可用.
E-mail: wuyh10@sei.pku.edu.cn

张颖(1983－),男,博士,讲师,主要研究领域为分布式系统,软件中间件,软件工程.
E-mail: zhangying06@sei.pku.edu.cn

黄罡(1975－),男,博士,教授,博士生导师,CCF 会员,主要研究领域为分布式系统,软件中间件,软件工程.
E-mail: hg@pku.edu.cn

张晓东(1989－),男,学士,主要研究领域为分布式系统,软件中间件,软件工程.
E-mail: zhangxd10@sei.pku.edu.cn

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件中间件,软件体系结构.
E-mail: meih@pku.edu.cn