异构云平台性能监控与分析研究

黄骁飞[1] 白晓颖[1] 苑丽杰[2]
(清华大学计算机科学与技术系 北京 100084)[1] (北京大学软件与微电子学院 北京 102600)[2]

摘 要 云计算是近年来兴起的一种新型的计算和服务模式,它采用多租户策略和按需付费计算模型,弹性地向用户提供看似无限的计算和存储资源。当前,云计算已经成为互联网环境下新型计算模式研究的焦点和热点。但是,云计算刚刚处于起步阶段,还没有形成统一的、公认的云计算标准。如何比较和评价多个不同公司、不同类型云平台的性能,为用户选择云平台提供依据,成为现在云计算性能评估的一个迫切需要解决的问题。针对以上问题,设计并构建了跨云平台的性能监控系统,该系统根据用户性能需求定义为多种云平台提供实时性能数据监控,进行跨云平台的性能比较分析,并且图形化地展示给用户。

关键词 云平台,异构性,性能监控,性能分析
中图法分类号 TP393 **文献标识码** A

Performance Monitoring and Analysis of Heterogeneous Cloud Platforms

HUANG Xiao-fei[1] BAI Xiao-ying[1] YUAN Li-jie[2]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[1]
(School of Software and Microelectronics, Peking University, Beijing 102600, China)[2]

Abstract Cloud computing is a new kind of computing and service model rising in recent years. By multi-tenant strategy and pay-on-demand model, it elastically provides users with seemingly unlimited computing and storage resources. Currently, cloud computing has become the focus and hotspot of new computing models in the Internet environment. How to compare and evaluate the performance of cloud platforms with different types from different companies to provide the basis for users to select one cloud platform, has become one of problems to be solved for cloud computing performance evaluation. To solve the above problem, a performance monitoring system to cross multiple cloud platforms was designed and implemented. It provides real-time performance data monitoring based on user requirements definition, comparative analysis of performance across cloud platforms, and displays the results to users graphically.

Keywords Cloud platforms, Heterogeneity, Performance monitoring, Performance analysis

1 引言

云计算是近年兴起的新型的计算和服务模式,它基于分布式计算、并行计算、网格计算和虚拟化技术,建立资源池,以按需付费的方式向用户提供硬件租赁、数据存储、计算分析和软件应用等不同类型的在线服务。根据NIST的定义[1,2],云计算可提供多个不同层次的服务类型,例如基础设施即服务IaaS、平台即服务PaaS和软件即服务SaaS。云计算已经成为互联网环境下,新型计算模式研究的焦点和热点。

云计算的最显著特征就是采用多租户策略和按需付费计算模型,弹性地向用户提供看似无限的计算和存储资源,提高资源的使用率并降低成本[3,4]。当前,各大云计算厂商如Amazon[5],IBM[6],Google[7],Microsoft[8],VMware[9]等公司都推出自己研发的云计算服务平台[10]。但是,由于云计算刚刚处于起步阶段,还没有形成统一的、公认的云计算标准。每一个云计算服务平台,均提出了特有的服务协议,采用不同的性能指标定义其服务质量,用户需要通过其提供的控制台、用户终端或者Open API访问其云平台服务。性能评估是用户选择云平台的依据,如何比较和评价多个不同公司、不同类型云平台的性能,是现在云计算性能评估的一个迫切需要解决的问题,跨云平台的比较分析在云计算的发展中愈发重要。

当前,许多云平台,例如Amazon AWS,都提供了在线性能监控服务,使得用户通过控制台或者Open API可以收集、分析、查看系统和应用性能数据以便更加快速和有效地做出操作和商业决定。

针对云平台的异构化特点,许多厂商都开发了跨云平台的管理工具,包括System Center Virtual Machine Manager[11]、vCenter Multi-Hypervisor Manager[12]、HotLink SuperVISOR[13]和openstack[14]。这些工具均提供了对于多种云平台及Hypervisor的管理功能,也包括了性能监控的功能,但是其性能监控仅仅是调用云平台的性能监控服务。

目前,云平台的性能监控服务仍然存在多方面的不足,主

到稿日期:2013-01-25 返修日期:2013-04-12 本文受国家自然科学基金(61073003),国家重点基础研究发展计划(973计划)(2011CB302505)和富士通实验室资助。

黄骁飞 男,硕士生,主要研究方向为软件工程、软件测试,E-mail:huangxfestudent@gmail.com;**白晓颖** 女,副教授,主要研究方向为软件工程、软件测试、软件项目管理等;**苑丽杰** 女,硕士生,主要研究方向为软件工程。

要表现在下面4个方面：(1)当前各个云平台只提供针对其自身的性能监控服务，缺乏跨云平台的性能监控和对比，不能满足在不同的云平台之间动态迁移系统所需要满足的性能监控需求；(2)每种云平台均使用自己定义的性能指标和监控策略，平台之间缺乏统一的性能需求定义和描述、数据采集和分析方法；(3)每种云平台均使用自己的Open API，缺乏接口标准为用户和开发者提供跨云平台的性能监控服务；(4)当前的性能监控均是对基本的性能指标(CPU、内存、硬盘、网络等)的监控，缺乏对其服务质量特性(如伸缩性)的评价模型。

针对以上问题，本文设计并构建了一个跨云平台的性能监控系统。该系统针对不同云平台的开放接口协议，定制适配器，在线实时采集、整合不同云平台的性能数据，对其进行分析和图形化展示，从而提供可跨不同云平台的实时性能监控、性能需求定义、描述、数据采集和分析的能力。

本文第2节描述了监控系统的体系架构；第3节详细叙述了跨云平台的适配器机制的实现；第4节简要介绍了云平台性能实时监控与分析比较；第5节展示了原型系统的实现细节；最后进行了总结。

2 体系架构设计

如图1所示，本文中将从以下几个方面构建跨云平台的性能监控系统。

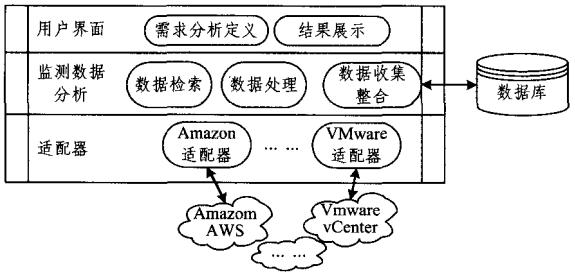

图1 监控系统体系架构

• 用户界面：提供性能监控需求定义的用户界面，并展示性能监控和分析结果。用户按照需求定义性能数据的监控策略，包括被监测的对象、被监测资源或事件类型，以及数据采集的类型、频率和范围等；按照用户需求收集的性能数据再以趋势图、图表等形式展示给用户，供用户对云平台性能进行分析和比较。

• 监测数据分析：将用户性能监控和分析需求转化为数据采集的查询请求，整合、分析所采集到的各类性能数据，根据数据的来源、类别、时效性归类分析各种数据，形成各类数据分析报告。

• 适配器：利用云平台的监控服务，针对不同的云平台开放接口协议，将查询请求转换成特定的消息格式，获取性能数据。

3 跨平台适配器设计

监控系统通过跨平台适配器机制消除多个云平台之间的异构性。图2是跨平台适配器的整体结构。

每个云平台的适配器通过开放接口从各个云平台获取云平台的基本信息以及性能数据，并将其转换成为监控系统定义的对象模型和统一性能模型。监控系统内的不同云平台间的数据就以对象模型和统一性能模型的形式进行交互。监测数据分析层通过适配器接口获取对象模型数据和统一性能模型数据，将性能数据以统一的性能数据模型形式存入数据库，并将对象模型和性能指标模型转换为XML描述语言形式提供给用户界面以展示给用户。

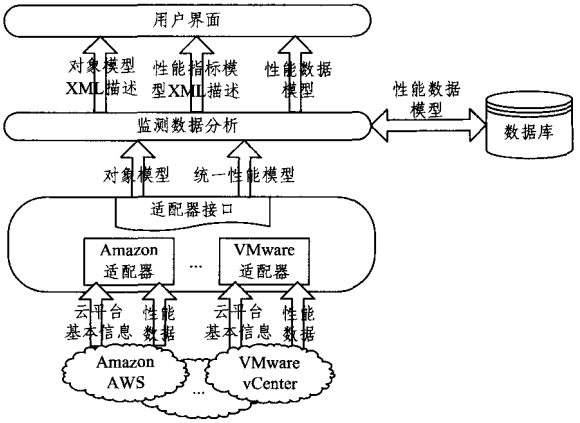

图2 跨平台适配器整体结构

3.1 对象模型

由于云平台的异构性特征，各个云平台在服务协议、访问方式、管理策略、性能指标等方面均有很大区别，监控系统将云平台基本信息的基本特征抽象，形成一套统一的对象模型，构成数据的元模型，映射到异构云平台上特定的数据结构。

抽象对象模型主要包括5种：*Cloud*、*Instance*、*Metric*、*Data*和*SLA*。图3给出了这5种对象模型的定义和关联关系。对象模型可以用元组的概念进行描述。

• $Cloud := \langle Spec, Instances, Metrics \rangle$，表示云平台的对象模型。

其中，$Spec := \langle name, url, username, password \rangle$，*name*是云平台的名称，*url*是访问平台的地址，*username*和*password*分别是登录云平台的用户名和密码；$Instances := \{Instance_i\}$是属于该云平台的所有服务实例；$Metrics := \{Metric_i\}$是该云平台提供的性能指标集合。

• $Instance := \langle Spec, Datas \rangle$，表示云平台向用户提供的服务实例的对象模型。

其中，$Spec := \langle name, Cloud, key \rangle$，*name*是该服务实例的名称，*Cloud*是该服务实例所属云平台，*key*表示所属云平台提供的该服务实例的识别信息；$Datas := \{Data_i\}$是云平台提供的该服务实例的性能数据。

• *Metric*是模型监控系统的性能指标模型，*Data*是监控系统的性能数据模型，下一节统一性能模型中会对它们专门进行介绍。

• $SLA := \langle Spec, Data, threshold \rangle$，是用户定义的服务水平协议模型。

其中，$Spec := \langle Name, Description \rangle$；*Data*是订立服务水平协议的数据对象；*threshold*是服务水平协议的监控阈值。

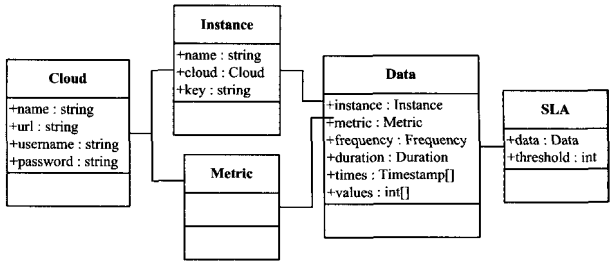

图3 对象模型定义

在各个云平台开放接口中，云平台基本信息基本上以对象的形式表示，都可以被抽象为上面设计的对象模型。表 1 以 Amazon EC2[5] 和 VMware vSphere[9] 为例，对比了它们的基本信息所对应的对象类。

表 1　Amazon EC2 和 VMware vSphere 对象模型[15,16]

云平台主要元素	VMware vSphere	Amazon EC2
Cloud	ServiceInstance	AmazonEC2Client
Instance	VirtualMachine	Instance
Metric	PerfCounterInfo	Metric
Data	PerfEntityMetric	Datapoint

3.2　统一性能模型

我们根据性能监控系统的功能需求，将各个云平台性能指标进行抽象，形成云平台的统一性能模型，以便对不同云平台同一资源的性能指标进行比较和分析。图 4 描述了监控系统的统一性能模型。

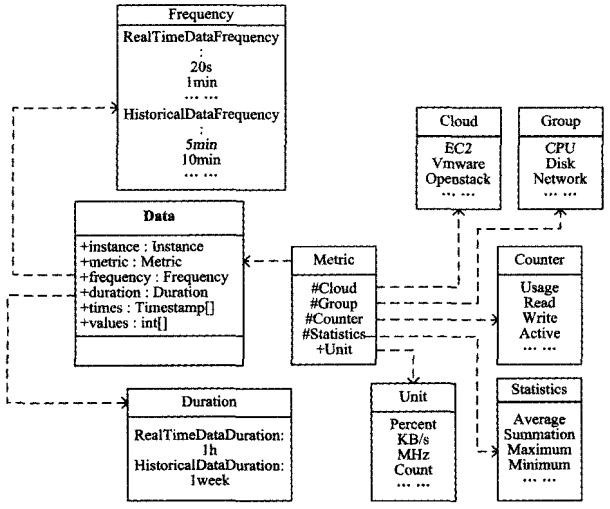

图 4　统一性能模型定义

统一性能模型包括性能指标模型和性能数据模型两部分。

性能指标模型可以表示为下面的五元组形式：*Metric* ::= ⟨*Cloud*, *Group*, *Counter*, *Statistics*, *Unit*⟩。

其中，*Cloud* 是该性能指标所属云平台的对象模型；每个云平台的性能指标按照所属资源类别不同分成若干 *Group*；每个 *Group* 中又可以有多种 *Counter*，*Counter* 主要描述该性能指标的特性，例如利用率(*Usage*)、读速率(*Read*)、写速率(*Write*)；*Statistics* 是指该性能指标的统计方法，例如平均数(*Average*)、最大值(*Maximum*)；*Unit* 是该性能指标的单位。

监控系统的每一个性能指标均可以用下面的四元组唯一定义：⟨*Cloud*, *Group*, *Counter*, *Statistics*⟩。举例来说，⟨EC2, CPU, Usage, Average⟩定义了 EC2 的 CPU 相关的一条性能指标，该性能指标统计的是 EC2 上 Instance 的 CPU 使用率的平均值。

性能数据模型的元组定义形式为：*Data* ::= ⟨*Instance*, *Metric*, *Frequency*, *Duration*, *times*, *values*⟩。

其中，*Instance* 表示所属服务实例对象；*Metric* 是性能数据所属的性能指标；*Frequency* 是性能数据采集频率；*Duration* 是性能数据保存时长；*times* ::= $\{time_i\}$，是数据采集的时间点集合；*values* ::= $\{value_i\}$，是性能数据值的集合。

监控系统的性能数据分为两种类型：实时性能数据和历史性能数据，通过参数选择的不同予以区分。实时性能数据是指最近某段固定时间长度内从各个云平台直接实时监控获取的性能数据，数据采集的时间间隔为该平台所能提供的最小时间间隔。用户如果希望能够更加灵活地定义性能数据的采集策略，可以自己定义历史性能数据。根据用户预先定义的采集频率和时间长度，系统对从云平台收集的实时性能数据进行计算和处理后，形成历史性能数据，并保存到数据库中。历史性能数据向用户提供可选择的更灵活的数据采集频率和更长的数据保存时长。表 2 以 Amazon EC2 和 VMware vSphere 为例，展示了实时性能数据与历史性能数据的采集频率和保存时长的参数选择实例。

表 2　实时性能数据与历史性能数据参数实例

	实时性能数据	历史性能数据
数据采集频率	20 秒(VMware) 60 秒(Amazon)	5 分钟，10 分钟，15 分钟，30 分钟
数据保存时长	1 小时	1 星期

图 5 展示了 EC2 的性能指标模型 XML 语言描述实例。

```
<?xml version="1.0"?>
- <Cloud label="EC2">
  - <Group label="CPU">
    - <Metric label="CPUUtilization">
        <Counter label="Usage"/>
        <Statistics label="Average"/>
        <Unit label="percent"/>
      </Metric>
    </Group>
  - <Group label="Disk">
    - <Metric label="DiskWriteBytes">
        <Counter label="Write"/>
        <Statistics label="Average"/>
        <Unit label="Bytes"/>
      </Metric>
    - <Metric label="DiskReadBytes">
        <Counter label="Read"/>
        <Statistics label="Average"/>
        <Unit label="Bytes"/>
      </Metric>
    - <Metric label="DiskWriteOps">
        <Counter label="NumberWrite"/>
        <Statistics label="Summation"/>
        <Unit label="count"/>
      </Metric>
    - <Metric label="DiskReadOps">
        <Counter label="NumberRead"/>
        <Statistics label="Summation"/>
        <Unit label="count"/>
      </Metric>
    </Group>
  - <Group label="Network">
    - <Metric label="NetworkIn">
        <Counter label="Received"/>
        <Statistics label="Average"/>
        <Unit label="Bytes"/>
      </Metric>
    - <Metric label="NetworkOut">
        <Counter label="Transmitted"/>
        <Statistics label="Average"/>
        <Unit label="Bytes"/>
      </Metric>
    </Group>
</Cloud>
```

图 5　Amazon EC2 性能模型 XML 示例

3.3　数据采集流程

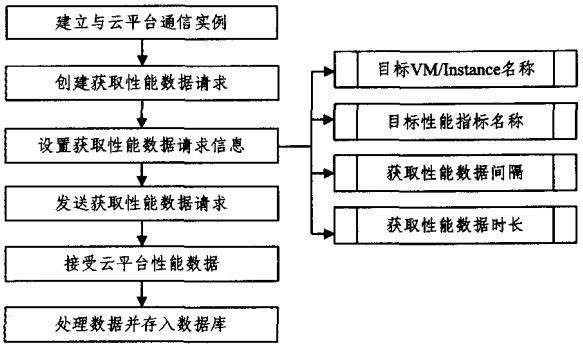

图 6　数据采集流程

不同云平台获取性能数据的流程以及需要调用的对象和方法也有很大差异。监控系统统一了各个云平台的数据采集流程，这样在进行数据采集的时候便屏蔽了云平台的种类，所

有的云平台均执行相同的数据采集流程。性能数据采集流程如图 6 所示。

表 3 对比了 Amazon EC2 和 VMware vSphere 云平台数据采集的具体流程实现。

表 3　Amazon EC2 和 VMware vSphere 数据采集流程[8,9]

数据采集流程	Amazon EC2	VMware vSphere
建立与云平台通信实例	建立 AmazonCloudWatchClient 实例	获得 PerformanceManager 实例
创建获取性能数据请求	建立 GetMetricStatisticsRequest 实例	建立 PerfQuerySpec 实例
设置获取性能数据请求信息	需要设置：Dimension、MetricName、Namespace、Peroid、StartTime、EndTime、Unit、Statistics	需要设置：目标 VirtualMachine、目标性能指标(Metric)、采样频率、采样数量
发送获取性能数据请求	调用 getMetricStatistics：AmazonCloudWatchClient 方法(参数是 GetMetricStatisticsRequest)	调用 queryPerf：PerformanceManager 方法(参数为 PerfQuerySpec)
接受云平台性能数据	返回 GetMetricStatisticsResult 实例	返回 PerfEntityMetricBase 实例
处理数据并存入数据库	调用 getDatapoints：GetMetricStatisticsResult 得到某个 Metric 的性能数据	解析 PerfEntityMetricBase 获得所需性能数据

3.4　适配器接口设计

监控系统使用接口模式(Interface)实现适配器机制(Adapter)。定义适配器接口，此接口向整个系统间接地提供云平台的性能监控服务，并且为监控系统集成的各个云平台实现适配器接口。适配器接口的具体描述如图 7 所示。

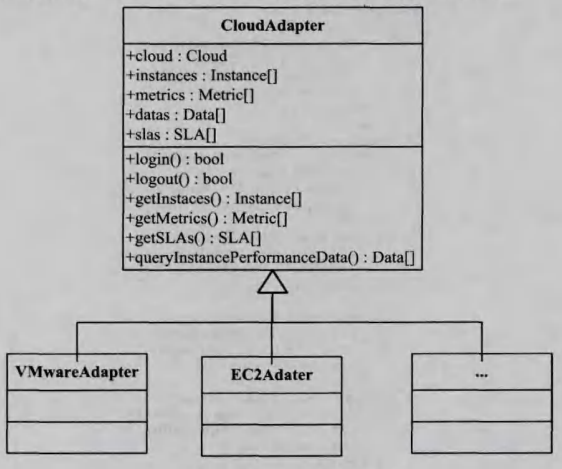

图 7　适配器接口设计

通过适配器机制，监控系统消除了云平台开放接口的异构性。由于为每个云平台均实现了对应的适配器，监控系统不再直接调用云平台的开放接口访问云平台提供的性能监控服务，而是通过访问适配器接口，间接地访问云平台的性能监控服务。这样便大大降低了监控系统与云平台开放接口之间的耦合性，将不同云平台的性能监控服务相统一，并且后续可以方便地向监控系统添加新的性能监控功能和集成新的云平台。

4　云平台性能实时监控与分析比较

监控系统通过对多个云平台获取的性能数据的处理、比较和分析，将监控系统的性能数据转化为可比较的统一形式，这样便可以用较为直观的量化指标对不同云平台服务的性能进行对比和评价，从而为用户选择满足自身需求的最合适云平台提供比较和依据。

4.1　数据处理

通过统一性能模型和性能数据模型的建立，所有通过开放接口获得的云平台的性能指标和数据均根据预先的定义和用户需求经过一些数学方法运算后处理为统一的形式供用户进行对比。监控系统的用户接口也提供了相应的接口，使得用户可以方便地比较多个云平台的同一性能指标。表 4 列举了处理数据的数学方法。

表 4　数据处理方法

数学方法	说明
单位换算	将不同云平台的同一性能指标的数据换算为同一单位
归一化	将性能数据换算成为相对值，即将绝对的物理数值转换为某种相对值关系
区间映射	把性能数据映射到[0,1]的小数区间内

4.2　多维度数据比较

监控系统从多个维度定义了性能数据模型，多维度包括性能数据的来源(所属云平台、所属服务实例)、性能指标类别(⟨*Cloud*,*Group*,*Counter*,*Statistics*⟩)以及数据采集策略(数据的采样频率、采样时间长度)等等。用户可以根据需求选择各个维度的参数，对它们进行动态组合，从不同的角度对比和评估云平台性能。

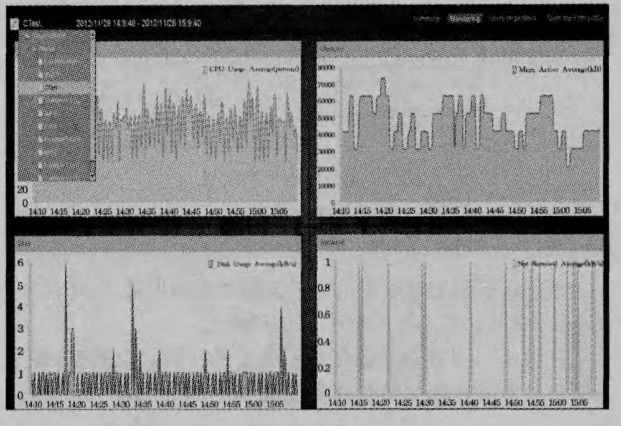

图 8　同一 Instance 多性能指标的实时性能比较

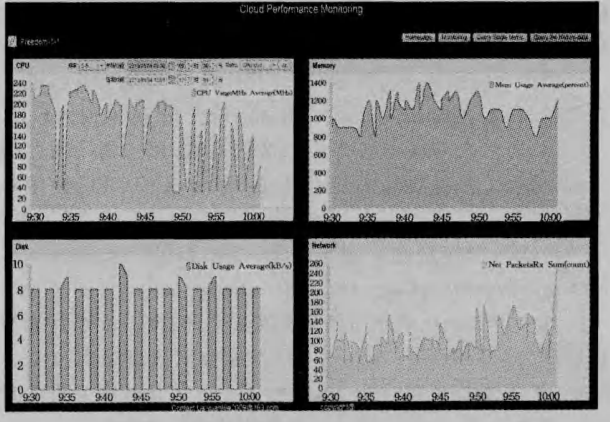

图 9　同一 Instance 多性能指标的历史趋势比较

目前，监控系统可以从 3 个维度比较性能数据，以供用户

进行性能对比和评估：(1)同一 Instance 多性能指标的实时性能比较；(2)同一 Instance 多性能指标的历史趋势比较；(3)多 Instance 同一性能指标数据比较。图 8—图 10 展示了这 3 种比较形式的界面截图。

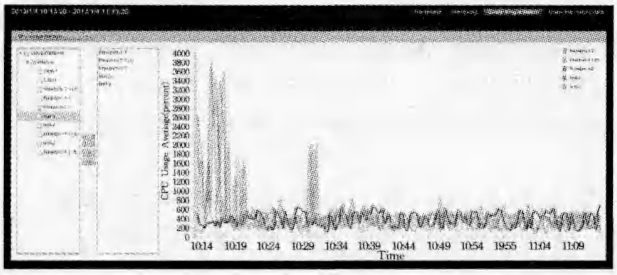

图 10　多 Instance 同一性能指标数据比较

5　原型系统实现

整个原型系统在 Ecplipse 平台上开发完成，Web 容器为 Apache Tomcat6.0，使用的编程语言是 Java，Web 界面语言使用的是 Flex；数据库使用的是 MySQL5.2，并且通过 Hibernate 构建持久化数据层；通过 VI SDK(VMware)和 Amazon AWS SDK(Amazon EC2)监控各个云平台、获取性能数据。

原型系统按照功能可以划分为 6 个功能模块：Adapter，Data Collector，Data Processor，Data Query，Requirement Analyzer和 User Interface。其体系结构如图 11 所示。

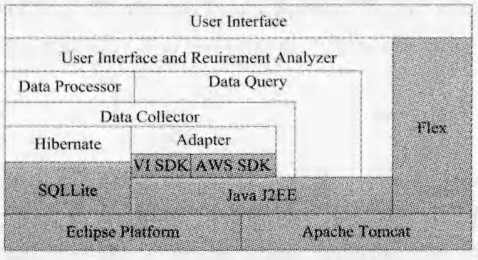

图 11　原型系统体系结构

• Adapter 模块：通过 Open API 监控各个云平台。Adapter 调用各个云平台自有的开放接口，监控云平台的运行状态，根据 Data Colloctor 的请求从云平台收集所需性能指标的实时监控数据，并返回给 Data Collector。

• Data Collector 模块：根据其他模块的请求，通过 Adapter或数据库获取所需的性能数据。Data Collector 会根据用户定义，将 Data Processor 处理完成后的性能数据存储到数据库中。Data Collector 模块是与 Adapter 和数据库交互的唯一模块。

• Data Processor：主要包括两个方面的功能：1)将实时性能数据转换为用户所要求的历史性能数据；2)将储存在数据库中的性能数据转换为 UI 模块所要求的数据，供 UI 模块进行图形化显示。

• Data Query：查询模块。处理和分析其他功能模块数据获取需求，从 Data Collector 获得相应的数据并返回给相应模块。

• Requirement Analyzer：从 UI 模块接受并解析用户需求，根据需求解析的结构调用相关的功能模块，然后接收功能模块返回的需求执行结果并将结果返回给 UI 模块。

• UI Interface：接收用户需求，并将根据用户需求所得到的性能数据图形化地展现给用户。

原型系统集成了两个典型云平台：Amazon EC2 和 VMware Infrastructure。用户可以访问其提供的开放接口，程序化地获取其定义的性能指标数据，并存储到数据库中，然后根据需求查询用户所需的性能数据，并且图形化地呈现给用户。

原型系统的性能指标集中在资源使用方面，主要包括 CPU、内存(Memory)、硬盘(Disk)和网络(Net) 4 种类型。表 5 展示了 Amazon EC2 和 VMware vSphere 的性能指标设置。

表 5　原型系统性能指标[17,18]

类型	性能指标	平台 EC2	平台 VMware	说明
CPU	Usage (percent)	√	√	当前使用的 CPU 资源比例
	UsageMHz (MHz)	×	√	当前正在使用的 CPU 资源
硬盘	Usage (kB/s)	×	√	硬盘读写的总速度
	Write (kB/s)	√	√	向硬盘写数据的速度
	Read (kB/s)	√	√	从硬盘读取数据的速度
	NumberWrite (count)	√	√	采样时间内写硬盘的次数
	NumberRead (count)	√	√	采样时间内读硬盘的次数
内存	Active (kB)	×	√	当前活跃 host 内存数量
	Consumed (kB)	×	√	虚拟机为 Guest OS 消耗的 host 内存数
	Usage (percent)	×	√	当前使用的内存比例
	Granted (kB)	×	√	映射给虚拟机的 host 内存量
	Shared (kB)	×	√	与其他虚拟机共享的内存量
	Overhead (kB)	×	√	虚拟机额外的内存消耗
网络	Usage (kB/s)	×	√	网络使用的总吞吐量
	Transmitted (kB/s)	√	√	网络上传的吞吐量
	Received (kB/s)	√	√	网络接收的吞吐量
	PacketsTx (count)	×	√	采样时间内上传的包数目
	PacketsRx (count)	×	√	采样时间内下载的包数目

结束语　本文主要介绍了一个跨云平台的性能监控系统。该监控系统利用云平台提供的 Open API，将两个不同的云平台的性能监控服务整合并集中，向用户提供了与平台无关的性能数据收集、分析和展示的功能，通过 Adapter 机制的实现，可以方便地在该监控系统中添加新的云平台。本文具体介绍了该系统的主要设计思路、模块组成以及工作流程。

未来的工作主要包括继续扩展监控系统：加入 SLA(服务水平协议)的定义和分析，通过多指标综合评估法建立用户服务质量模型，定性/定量刻画云平台服务性能对于用户需求的满意程度，为用户选择云平台服务提供一定的参考。

参 考 文 献

[1] Mell P, Grance T. The NIST Definition of Cloud Computing Version 15[M]. White Paper, V15. 2009

[2] 张建勋，古志民，郑超. 云计算研究进展综述[J]. 计算机应用研究，2010，27(2)：429-433

[3] Armbrust M, Fox A, Grith R, et al. Above the clouds: A Berkeley View of Cloud Computing[R]. CB/EECS-2009-28. Berley, USA: Electrical Engineering and Computer Sciences, University of California at Berkeley, 2009

[4] Vaquero L, Rodero-Marino L, Caceres J, et al. A break in the clouds: towards a cloud definition[J]. SIGCOMM Computer Communication Review, 2009, 39(1): 50-55

[5] Amazon Web Services[EB/OL]. http://aws.amazon.com/

[6] IBM Cloud Computing[EB/OL]. http://www.ibm.com/cloud-computing/us/en/

(下转第 163 页)

5.2 同步过程

1)反馈。依次对应用 1 和应用 2 执行对领域的反馈操作,反馈结果见图 8 最上面的领域特征模型。对于应用 1 中添加的两个特征:Reomte_Control 和 Internet,领域将其吸收为可选特征,并吸收特征依赖 Reomte_Control requires Internet;对于特征 Keypad 被替换为特征 FingerPrint,领域将其变为一个多选一特征组 Device_Type,包含两个变体:Keypad 和 FingerPrint。对于应用 2 添加的特征 Ambient,领域将其吸收为多选多特征组 Pre_defined_Value 的一个新的变体;而对于删除特征 Fire_Sprinkler,领域认为其是应用的一个特定需求而保持不变。

2)同步。对于应用 1,除了自身反馈引发的领域变化,领域还新添加了特征 Ambient、Windows_Management 及其子特征,应用 1 对其进行了定制,见图 8 中间应用 1 的特征模型。对于应用 2,除了自身反馈引发的领域变化,领域新添加了特征 Reomte_Control、Internet、特征 Windows_Management 及其子特征,此外特征 Keypad 变为多选一特征组 Device_Type,应用 2 对这些特征进行了定制,见图 8 最下面应用 2 的特征模型。至此,Smart_Home 的领域与应用同步完成,重新恢复了一致性。

结束语 本文对产品线领域和应用特征模型的演化场景进行了分析,提出了一个特征模型演化同步框架,其中包含针对不同种类特征模型描述提出的一种通用特征模型描述,以及领域和应用特征模型演化同步的规则。不同种类的特征模型只需提供自己与通用特征模型的 XSLT 转换代码,就可以共用文中框架的演化同步规则,实现领域和应用特征模型的一致性演化。文中 Smart_Home 的例子验证了框架的可用性。

当前的演化同步规则没有涉及到演化中可能存在的冲突问题(如多个应用对同一个特征进行了演化),接下来我们会对同步规则进行完善,以涵盖这些冲突的情况,并用大量的实例对框架进行进一步验证。

参 考 文 献

[1] Clements P C, Northrop L. Software Product Lines: Practices and Patterns[M]. Boston: Addison-Wesley, 2001

[2] Kang K C, Cohen S G, Hess J A, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study[R]. Pittsburgh: Carnegie-Mellon University Software Engineering Institute, 1990

[3] Kang K C, Jaejoon L, Donohoe P. Feature-oriented product line engineering[J]. Software, IEEE, 2002, 19(4): 58-65

[4] Peng X, Shen L, Zhao W. An Architecture-based Evolution Management Method for Software Product Line[C]// Proceedings of the 21st International Conference on Software Engineering, 2009. Boston: Knowledge Systems Institute Graduate School, 2009: 135-140

[5] Hwan C, Kim P, Czarnecki K. Synchronizing cardinality-based feature models and their specializations[C]// Proceedings of the First European conference on Model Driven Architecture-foundations and Applications, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 331-348

[6] Gamez N, Fuentes L. Software product line evolution with cardinality-based feature models[C]// Proceedings of the 12th international conference on Top productivity through software reuse, 2011. Berlin, Heidelberg: Springer-Verlag, 2011: 102-118

[7] Batory D. Feature models, grammars, and propositional formulas [C]// Proceedings of the 9th international conference on Software Product Lines, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 7-20

[8] Czarnecki K, Helsen S, Ulrich E. Staged Configuration Using Feature Models[C]// Software Product Lines, Third International Conference, SPLC, 2004. Boston, USA: Springer-Verlag, 2004: 266-283

[9] Benavides D, Trinidad P, Ruiz-Cort E S A. Automated reasoning on feature models[C]// Proceedings of the 17th international conference on Advanced Information Systems Engineering, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 491-503

[10] W3c. XSL Transformations (XSLT) Version 2. 0[S/OL]. http://www.w3.org/TR/xslt20/, 2007-01-23

(上接第 151 页)

[7] Google app engine[EB/OL]. http://appengine.google.com/

[8] Microsoft azure[EB/OL]. http://www.microsoft.com/azure/

[9] VMware Virtualization[EB/OL]. http://www.vmware.com/

[10] 刘鹏. 云计算[M]. 北京:电子工业出版社, 2010

[11] System Center Virtual Machine Manager [EB/OL]. http://www.microsoft.com/zh-cn/server-cloud/system-center/virtual-machine-manager.aspx

[12] VMware vCenter Multi-Hypervisor Manager[EB/OL]. http://www.vmware.com/support/mhm/doc/vcenter-multi-hypervisor-manager-10-release-notes.html

[13] HotLink SuperVISOR[EB/OL]. http://www.hotlink.com/

[14] openstack[EB/OL]. http://www.openstack.org/

[15] AWS SDK for Java Developer Guide [K]. Amazon Web Services, Inc. 2012

[16] Jin S. VMware VI and vSphere SDK: Managing the VMware Infrastructure and vSphere 1st[M]. Prentice Hall PTR Upper Saddle River, NJ, USA, 2009

[17] Performance Counters[Z]. VMware, Inc. Nov, 2008

[18] Amazon Cloud Watch Developer Guide [K]. Amazon Web Services, Inc. 2013