基于PaaS的云应用软件部署环境的元模型

刘欢欢 麻志毅 陈泓婕

(北京大学信息科学技术学院软件所北京大学高可信软件技术教育部重点实验室 北京 100871)

摘 要 PaaS是云计算的一种服务模式,用于提供应用程序容器服务。在PaaS上部署云应用软件的主要方式是调用API和编辑配置文件,这需要很大的学习成本并且容易出错。不同PaaS的API和配置文件有不同的语法,在PaaS上很难进行应用的迁移以及跨平台或多平台部署。提出了基于PaaS的云应用软件的部署环境的元模型,它降低了学习成本,使得部署过程更加自动化,简化了应用迁移,使跨平台或多平台部署成为可能。

关键词 PaaS,部署,元模型,建模,模型驱动

中图法分类号 TP311.5 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.10.011

Meta-model of PaaS-based Cloud Application's Deployment Environment

LIU Huan-huan MA Zhi-yi CHEN Hong-jie

(Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

Abstract PaaS is one of the service paradigms of cloud computing, which is used to provide the application container service. Calling API and editing configuration files are the main way of cloud application deployment in PaaS, which needs a lot of learning costs and are error-prone. API and configuration files of different PaaS have different syntax, as a result, application migration on PaaS is very difficult and cross-platform or multi-platform deployment is scarcely possible. This article proposed the meta-model of PaaS-based cloud application's deployment environment, which can lower the learning costs, make the deployment process more automated, simplify application migration, and make cross-platform or multi-platform deployment possible.

Keywords PaaS, Deployment, Meta-model, Modeling, Model-driven

1 引言

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并提高其可用性的模式。云计算包含3种服务模式,分别是软件即服务(SaaS)、平台即服务(PaaS)、基础设施即服务(IaaS)[1]。

NIST将PaaS这种服务模式定义为:开发者在云基础架构之上部署应用软件,这些应用软件使用了供应者支持的语言、库、服务和工具。开发者并不管理或控制底层的云基础架构,包括网络、服务器、操作系统、存储,但是能够控制部署到平台上的应用软件,并且能够配置应用软件运行的主机环境[1]。结合产业界对PaaS的实现(如Google App Engine),可以认为PaaS平台就是应用程序容器服务。

在传统计算平台上,部署环境是一组互相协作以提供一个托管软件模块的运行环境的计算资源的集合,包括配置集群、服务器和中间件等[2]。在PaaS平台上,部署环境除了包含预定义的运行环境,还包含用来配置应用和运行环境的部署工具。

为了部署云应用软件,PaaS平台提供了两类部署工具:1)通过调用API进行部署操作;2)通过编辑配置文件设置参数。API通常以WSDL、REST等形式提供,云供应商一般还提供封装了API的网页控制台、封装了API的命令行工具以及以代码库的形式提供的调用接口等,这些工具可以直接使用,也可以嵌入到应用程序中。而配置文件的参数既可使用文本编辑器编辑,也可通过控制台的配置界面进行设置。通常,结合命令行工具和配置文件即可在PaaS平台上部署应用。

事实上,API和配置文件等部署工具是一种领域特定语言(DSL),它们都有自己的具体语法。然而,这些API和配置文件的具体语法并不简单,各个PaaS的具体语法也各不相同,需要很大的学习成本才能学会这些语法,并且采用直接调用API和编辑配置文件的方式部署应用很容易出错。开发人员在选择PaaS平台时或许需要试用多个平台,学习多个平台的部署环境的具体语法显然是耗时又低效的,所以开发人员需要一种抽象语法,使得能够使用一致的语法形式描述不同PaaS平台的部署环境。

目前对PaaS上应用部署的研究主要集中在自动化部署、多平台部署和跨平台部署等方面。虽然通过构造一种新的领域特定语言[6,7]能够使得部署过程自动化,但是对不同的PaaS分别构造领域特定语言是不切实际的。通过在不同PaaS平台之上构造一个统一的抽象层[12,13]虽然能够在一定程

到稿日期:2014-10-25 返修日期:2015-03-09 本文受国家自然科学基金(61272159),北京市自然科学基金(4122036)资助。

刘欢欢 男,硕士生,主要研究方向为移动应用开发技术、云应用开发技术等,E-mail:liuhuanhuan@pku.edu.cn;麻志毅 男,副教授,主要研究方向为软件建模、智能化软件开发方法与技术等;陈泓婕 女,副教授,主要研究方向为软件建模、智能化软件开发方法与技术等。

度上掩盖不同平台的差异，但是很难面对 PaaS 平台的改变。

元模型是一种符合要求的抽象语法。通过从各个 PaaS 平台的 API 语法参考和配置文件模式等具体语法中提取出部署环境的元模型，学习成本将会大大降低，部署过程也将会得到很大简化。本文根据目前在 PaaS 平台上部署应用存在的困难，做了两部分工作：1）从 PaaS 平台的部署 API 语法参考和配置文件模式等代表性的具体语法中提取出以元模型表示的抽象语法；2）阐述了一种模型驱动的部署方法，该方法需要元模型支持。

部署环境的元模型和模型驱动的部署方法至少能在以下 3 方面对云应用软件的部署提供帮助：1）通过从说明文档等具体语法中提取出各个 PaaS 平台部署环境的平台相关元模型，有助于对 PaaS 平台部署环境的理解；由于各个 PaaS 平台存在一定的相似性，还能够从平台相关元模型中归纳出平台无关元模型。2）使用元建模技术[14]，能够由平台相关元模型生成相应的部署建模工具，使用部署工具建立平台相关的部署模型，并由部署模型生成部署命令行和配置文件，就能实现自动化部署。3）由平台无关的元模型生成相应的部署建模工具，使用部署工具建立平台无关的部署模型，并根据目标 PaaS 平台转换为平台相关的部署模型，就能实现跨平台或多平台部署。

本文第 2 节提取了 Amazon、Microsoft 和 Google 3 个公司的 PaaS 平台的部署环境的平台相关元模型；第 3 节对 3 个平台相关的元模型进行了对比和分析，并归纳出了平台无关的元模型；第 4 节列举了一系列应用场景以说明本文工作的意义；第 5 节讨论了相关工作；最后总结全文并展望未来。

2 部署环境的平台相关元模型

2.1 建立元模型的过程

建立部署环境元模型的过程分为图 1 所示的 2 个步骤。首先提取出 3 个典型的 PaaS 平台部署环境的平台相关元模型，然后归纳出平台无关元模型。

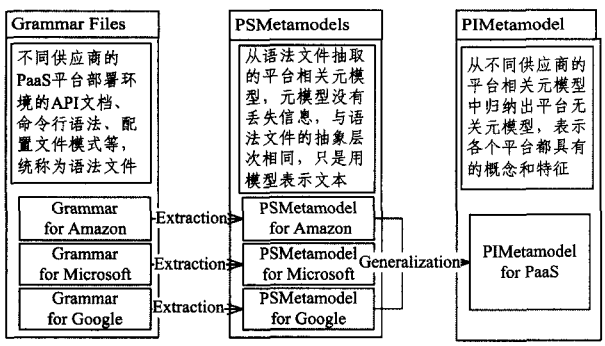

图 1 PaaS 部署环境元模型的提取和归纳

本文选择了 3 个典型的 PaaS 平台建立部署环境的平台相关元模型，分别是：AWS Elastic Beanstalk、Azure Cloud Services 和 Google App Engine。

PaaS 使用开发指南、API 语法参考、命令行参考、配置文件模式等指导部署，本文称这些文件为 PaaS 平台部署环境的具体语法。平台相关元模型并没有丢失语法信息，与具体语法相比，元模型只是以模型形式代替了文本形式表示的语法信息。

平台无关元模型提供了一种简明的方式描述在 PaaS 平台上部署应用的通用过程。由于这些通用的部署过程是多个 PaaS 平台共有的特征，因此相比平台相关元模型，平台无关元模型丢失了部分语法信息。部署实践中，将平台无关的部署模型转换为平台相关的部署模型时，还要增加一些平台特定的信息。

鉴于篇幅有限，本节只阐述 3 个 PaaS 的部署环境平台相关元模型的包的构成及划分，具体的元模型见文献[3-5]。

2.2 平台相关的元模型

1. AWS Elastic Beanstalk

AWS Elastic Beanstalk 是亚马逊的 PaaS 平台，提供了一种在 AWS 云中简单、快速部署和管理应用程序的 PaaS 解决方案。只需上传应用程序，Elastic Beanstalk 将自动处理有关容量预配置、负载均衡、自动扩展和应用程序运行状况监控的部署细节。本文对 AWS Elasitc Beanstalk 的部署环境元模型作如图 2 的划分，元模型的详细信息由文献[3]给出。

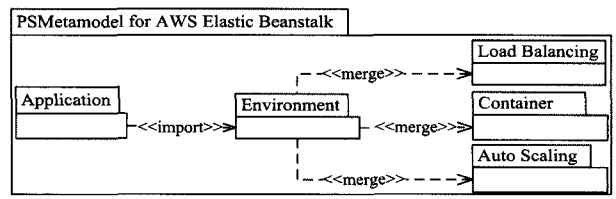

图 2 AWS Elastic Beanstalk 部署环境元模型的划分

2. Azure Cloud Services

Azure Cloud Services 是微软的 PaaS 平台。使用 Azure Cloud Services 能够快速部署和管理功能强大的应用程序和服务。只需要上传应用程序，Azure 即可处理部署细节（从设置和负载平衡到运行状况监视）以实现持续可用性，开发者只需专注于应用程序而非基础结构。本文对 Azure Cloud Services的部署环境元模型作如图 3 的划分，元模型的详细信息由文献[4]给出。

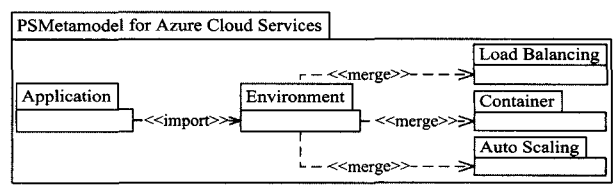

图 3 Azure Cloud Services 部署环境元模型的划分

3. Google App Engine

Google App Engine 是 Google Cloud Platform 的 PaaS 平台。在 App Engine 上可以使用内置服务轻松地开发应用，提高生产率。App Engine 是基于谷歌可靠的数据中心的、完全由谷歌管理的平台，用户只需上传应用，App Engine 负责基础设施管理。本文对 Google App Engine 的部署环境元模型作如图 4 的划分，元模型的详细信息由文献[5]给出。

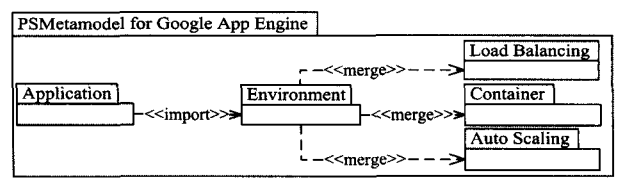

图 4 Google App Engine 部署环境元模型的划分

3 部署环境的平台无关元模型

3.1 3 个 PaaS 平台的异同

在详细分析了亚马逊、微软和谷歌的 PaaS 平台后，本文

发现了这些平台的相似之处：平台内置了负载均衡、自动伸缩和性能监视等服务，并基于这些服务构建了完整的运行环境；运行环境包含一个实例的集群，其中所有实例拥有相同的配置——由 CPU 和内存等硬件性能参数描述的实例等级、由操作系统和应用容器描述的软件堆栈，可以在运行时进行配置。

3 个供应商的 PaaS 平台也有许多不同的地方。其一是相同类型服务的细节差别很大。其二是内置服务和外部服务的划分不同，比如在 Google App Engine 中，队列服务和缓存服务是平台内置的服务；而在 AWS Elastic Beanstalk 和 Azure Cloud Services 中队列服务和缓存服务是外部服务。

本文总结了 PaaS 平台的通用部署环境，并对部署环境的平台无关元模型作如图 5 所示的划分。

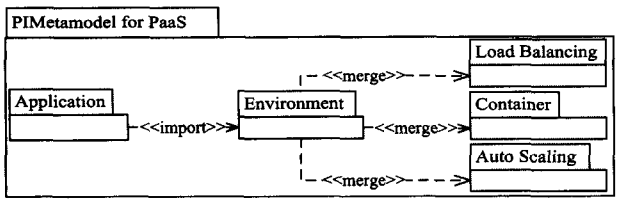

图 5　PaaS 部署环境平台无关元模型的划分

对于元模型的每一部分，本文将综合第 3 节中 3 个平台的元模型给出。总体遵循如下策略：3 个平台中 2 个或 3 个平台出现的类将出现在本节的元模型中。由于该部署环境的通用性，本节给出的元模型就是 PaaS 平台部署环境的平台无关的元模型。

3.2　平台无关的部署环境元模型

1. Application 包：应用结构

应用结构（Application）的元模型支持对 PaaS 应用及其组成部分的描述，如图 6 所示。

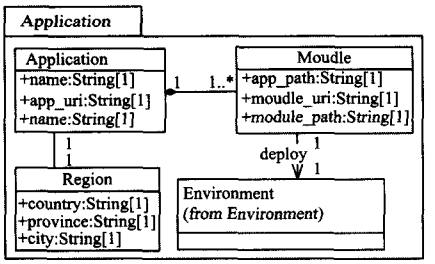

图 6　Application 包

Application 类表示部署到 PaaS 平台上的云应用。name 指定在云平台上创建的项目的名称。app_uri 指定在云平台上定位该应用的标识符，并生成访问该应用的 URL：app_uri. PaaSPlatformURL. com。app_path 指定应用的源代码的路径，在部署整个应用时要指定该路径。一个应用包含一个或多个模块。

Region 类表示将应用部署至某个区域。Moudle 类表示应用的一个逻辑组成部分。module_uri 指定在一个 PaaS 应用中定位该模块的标识符。一个模块部署到一个特定的运行环境中。

2. Environment 包：运行环境

运行环境（Environment）的元模型支持对运行 PaaS 应用的一个模块的环境的硬件性能和软件堆栈的配置，如图 7 所示。

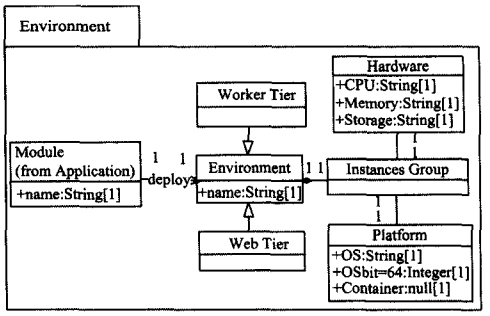

图 7　Environment 包

Environment 类表示 PaaS 应用的一个模块运行所需的软硬件资源。应用的每个模块都要部署到各自的运行环境中。运行环境根据其上运行的程序或脚本的生命周期的不同，分为 Web 层运行环境和 Worker 层运行环境 2 类，前者运行前端程序，后者运行后端程序。

Instances Group 类表示运行环境拥有的实例集合。实例集合中的每个实例都拥有相同的软硬件资源。

Hardware 类表示实例的硬件性能的配置。使用 CPU 频率、内存大小和存储容量（可选）定义虚拟机的性能。

Platform 类表示实例的软件堆栈的配置。使用操作系统、操作系统支持的总线宽度（32Bit/64Bit）和容器类型来定义软件堆栈。

3. Load Balancing 包：负载均衡

负载均衡（Load Balancing）的元模型支持对负载均衡器及其粘性会话和健康检查策略的配置，如图 8 所示。

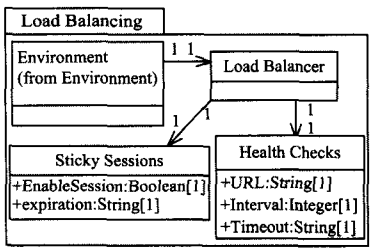

图 8　Load Balacing 包

Load Balnacer 类表示负载均衡器。PaaS 平台提供了默认的负载均衡器，不同的具体 PaaS 平台以不同的方式支持对复杂均衡器的配置。

Sticky Sessions 类表示负载均衡中的粘性会话策略。可以设置是否启用该策略以及会话的超时时间。

Health Checks 类表示负载均衡中的健康检查机制。需要设置健康检查时请求的 URL、请求的超时时间和请求的间隔。

4. Auto Scaling 包：自动伸缩

自动伸缩（Auto Scaling）的元模型支持对运行环境实例数量伸缩方式的选择和配置，如图 9 所示。

Scaling Type 类表示 PaaS 平台提供的伸缩方式。PaaS 平台一般会提供 2 种伸缩方式：手动伸缩和自动伸缩。

Manual Scaling 类表示手动伸缩。这种伸缩方式较简单，只需要设置固定数量的实例，这些实例构成了运行环境的实例集合，启动后一直处于运行状态。

Automatic Scaling 类表示自动伸缩。这种伸缩方式需要设置实例数量的最小值和最大值、每次扩展时增加的实例数量、每次收缩时减少的实例数量。自动伸缩一般还要设置触发机制。

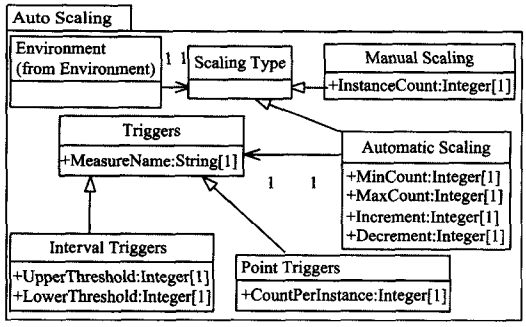

图 9 Auto Scaling 包

Triggers 类表示自动伸缩操作的触发机制。触发机制首先需要指定一个测量指标，根据该指标的数值决定是否触发伸缩动作。测量指标有多种类型，可以是 CPU 使用率、网络流量、内存使用率等性能指标，也可以是请求数量、队列中消息数量等负载指标。根据指标类型的不同，触发机制也可分为两种：区间触发和单点触发。

Interval Triggers 类表示区间触发。这种触发机制指定一个上界阈值和一个下界阈值。当测量指标的数值超过上界时，触发扩展动作；当测量指标的阈值低于下界时，触发收缩操作。

Point Triggers 类表示单点触发。这种触发机制指定单个实例能够承受的负载。需要的实例数量为测量指标的总负载除以单个实例的负载，并根据此数量的变化触发伸缩操作。

5. Container 包：容器

容器(Container)的元模型支持配置运行时实例的应用服务器的容器。对容器的配置适用于同一个实例集合中的所有实例，如图 10 所示。

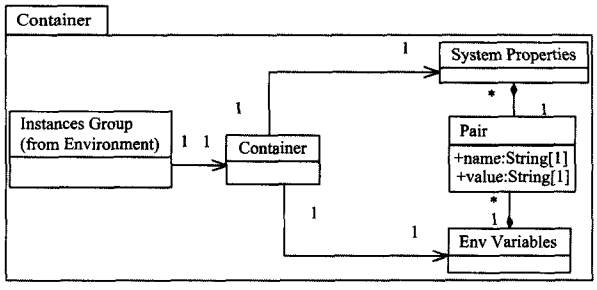

图 10 Container 包

各个 PaaS 平台对容器配置的支持几乎没有共同点。唯一的共同点是可以以键值对的形式设置多个系统属性和环境变量。

4 应用场景

4.1 模型驱动的部署

基于 PaaS 部署环境元模型，本文提出了模型驱动的部署方法，如图 11 所示。

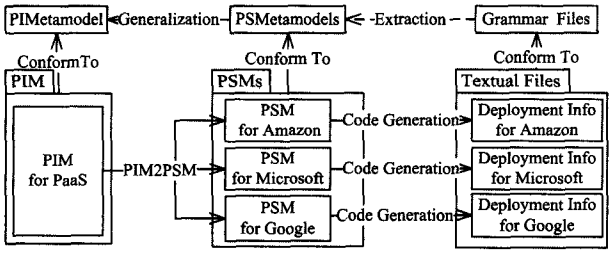

图 11 模型驱动的部署方法

部署过程分为 3 个阶段：建立平台无关的部署模型、将平台无关部署模型转换为目标平台的平台相关部署模型、由平台相关部署模型生成部署文本。本文以部署一个简单的 Java 应用来说明模型驱动的部署。

1)平台无关部署模型。使用由平台无关的元模型生成的部署建模工具，可以建立平台无关的部署模型。该部署示例的平台无关部署模型如图 12 所示。

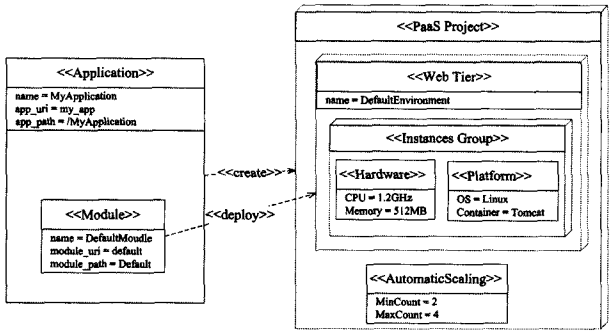

图 12 平台无关的部署模型

2)平台相关部署模型。使用模型转换工具，定义好相应的转换模型，就能把平台无关的部署模型转换为目标 PaaS 平台的平台相关部署模型。使用由平台相关的元模型生成的建模工具，能够直接建立平台相关的部署模型，或者对转换得到的平台相关的部署模型进行调整。以部署到 AWS Elastic Beanstalk 为例，部署示例的平台相关部署模型如图 13 所示。

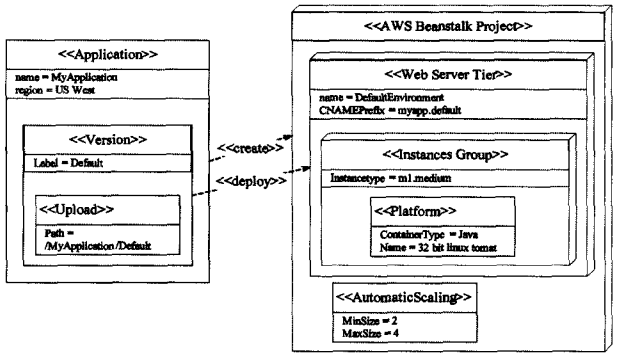

图 13 平台相关的部署模型

3)部署文本。使用代码生成工具，定义好生成代码的模板，就能由平台相关的部署模型生成部署文本。

4.2 快速理解部署环境

本文的元模型提取自 API 参考、配置文件模式定义、开发指南等说明文档，元模型提供了对部署环境直观、明确的描述。通过查看元模型，开发人员能够对 PaaS 平台提供的服务和 PaaS 平台对应用结构的要求有较好的理解，避免了直接查看复杂的说明文档。各个 PaaS 平台的元模型都是以相似的方式组织，因此便于对多个 PaaS 平台进行对比。

例如，通过参考文献中的普通相关的元模型，开发人员能够明确谷歌 PaaS 平台中的应用可以进行两层逻辑构成划分，而亚马逊和微软的 PaaS 平台中的应用只能进行一层划分；亚马逊和微软的负载均衡中提供了类似的健康检查策略，而谷歌并没有提供该策略；亚马逊和微软提供了类似的自动伸缩功能，而谷歌的自动伸缩受限于运行环境的类型等。

从模型获取知识的速度要比从文本快很多。通过快速理

解部署环境，开发人员能迅速对比多个 PaaS 平台的优缺点，选择一个适合的平台。

4.3 逆向工程

对于已经部署到某个 PaaS 平台上的应用，其已有的部署文本已经很好地描述了对运行环境的要求。但是如果想将该应用部署到其他 PaaS 平台，就避免不了学习新的部署环境、创建新的部署文本。

通过逆向工程，能够从已有的部署文本中抽取出平台相关的部署模型，并转换为平台无关的部署模型。得到平台无关的部署模型以后，只要做少量的调整，就能生成新的目标 PaaS 平台的部署文本。

逆向工程有助于不同 PaaS 平台之间的应用迁移，降低了平台锁定的风险。另外，对运行环境的配置也可以作为一种模板应用于另外一个应用程序，使得另外一个应用程序使用与已部署的应用相同的运行环境。将部署文本提取为部署模型，避免了多次配置相似的运行环境，可重复利用部署配置。

4.4 多平台部署

随着 PaaS 平台种类的增多，企业可能会在多个 PaaS 平台上分别部署多个相互协作的应用，以充分利用不同 PaaS 平台的优点。

例如 Google App Engine 提供了电子邮件服务，但是提供的负载均衡服务不能进行详细定制；而 AWS Elastic Beanstalk 没有提供邮件服务，但是负载均衡服务很完善。如果一个企业应用需要集成一个邮件服务，同时也需要强大的负载均衡，特别是有对健康检查策略可配置性的要求，那么将该企业应用的不同功能模块分别部署到 2 个 PaaS 平台上可能会是一种比较好的解决方案。

如果能够在多个 PaaS 平台上方便地部署一组协同工作的应用，那么能够使用的 PaaS 平台的功能就不局限于一个单独的 PaaS 平台，而是多个 PaaS 平台的集合。

模型驱动的部署方法为在多个 PaaS 平台上部署一组应用提供了可能。通过以统一的方式建立一组应用的平台无关的部署模型，提供了这组应用的全局部署视图，只要为其中的每个应用选定目标 PaaS 平台，就能很快完成部署。

当然，跨平台部署的目标也是能够达成的。事实上模型驱动的部署过程与生俱来就是支持跨平台的，平台无关模型隐藏了不同平台的细节。

5 相关工作

自动化部署。PaaS 平台提供了命令行脚本和配置文件的方式进行应用部署，为自动化部署提供了可能。Neptune[6] 是一个简单的基于 Ruby 的领域特定语言，使得在不同云计算系统上对科学软件框架的配置和部署更加自动化，并集成了对 MPI、MapReduce、UPC 等软件框架的支持。Eirik Brandtzæg[7] 等人提出使用基于构件的方法对云应用建模，利用现有的部署描述符开发了一种高层次的领域特定语言，并通过部署一个原型应用阐述了该领域特定语言的用法。Johannes Wettinger[8] 等人认为在 PaaS 平台上部署应用既需要在平台上部署应用本身，也可能需要部署应用需要的中间件组件，为此提出了一个面向中间件的方法用于在一定程度上定义中间件组件的部署，以提供一个定制化的 PaaS 平台，并使得在其上的应用部署自动化。

多平台部署。分布式的多平台云应用需要同时部署到多个 PaaS 平台，但是由于 PaaS 平台的多样化和异构性，这样的多平台部署十分困难。Cloud4SOA[9] 是一个跨越多个异构 PaaS 平台的多平台云应用管理工具，支持多个异构平台的语义互操作性。Saloon[10] 框架使用了基于特征模型的模型驱动方法，支持对多个异构云平台的配置。Nicolas Ferry[11] 等人阐述了一种模型驱动的技术和方法，用于促进多平台应用的资源预配置、部署、监视和适应等的规约。

跨平台部署。跨平台的云应用可以不加修改地部署到多个 PaaS 平台中的任意一个平台，但是由于平台的异构性，这样的部署往往需要额外的工具支持。Binh Minh Nguyen[12] 等人提出了一个简化云应用的开发和部署的抽象层，该抽象层隐藏了云平台的底层细节，在该抽象层上开发的应用可以部署到任何一个具体云平台。Mohamed Sellami[13] 等人提出了一个统一描述模型支持独立于目标 PaaS 平台表示应用，并且给出了一套通用的 PaaS 应用的资源预配置和服务管理 API。

结束语 本文建立了 PaaS 平台部署环境的元模型，元模型分为平台相关的元模型和平台无关的元模型。本文分别为亚马逊、微软和谷歌公司的 PaaS 平台建立了平台相关元模型，这些元模型提取自 API 参考、模式定义等说明文档。通过归纳这 3 个平台相关元模型的共同点，建立了平台无关的部署模型。本文还阐述了一种模型驱动的部署方法，并介绍了一些基于该方法的应用场景。

本文工作仍有一些有待完善的地方，今后的工作主要有以下几个方面：1）元模型还不够准确，需要进一步调整。由于本文工作任务量大、时间紧张，加之 PaaS 平台的说明文档也在不断更新，本文的元模型还有一些不准确的地方。2）建立更多平台的元模型。为了提高本文方法的适用性，还需要提高对更多 PaaS 平台的支持，为此需要建立更多元模型。3）设计相应的建模工具提供建模支持。MA Zhiyi[14] 等人阐述了基于元建模构建建模工具的技术，结合本文的元模型就能够自动生成用于部署建模的建模工具。此外还需要选取合适的模型转换和代码生成工具支持相应的建模流程。

参 考 文 献

[1] Badger M L, Grance T, Patt-Corner R, et al. Cloud Computing Synopsis and Recommendations[R]. NIST SP-800-146, 2012

[2] IBM. Deployment environments [OL]. http://pic.dhe.ibm.com/infocenter/dmndhelp/v7r5m1/index.jsp?topic=%2Fcom.ibm.wbpm.ref.doc%2Fhelp_nd%2Findex.html

[3] Platform specific meta-model of deployment environment of AWS Elastic Beanstalk [OL]. http://download.csdn.net/download/u012261044/8386785

[4] Platform specific meta-model of deployment environment of Azure Cloud Services[OL]. http://download.csdn.net/download/u012261044/8386813

（下转第 80 页）

索 HDFS 空闲块的时间随着元数据的减少而减少。

结束语 HDFS 是为存储大文件而设计，当存储大量小文件时会引起 I/O 性能瓶颈，同时，Namenode 主存里的元数据也会迅速增长，造成瓶颈。本文提出的合并及缓存策略将大量关联小文件合并成大文件后存储于 HDFS 上，有效缓解了 Namenode 主存的瓶颈问题；通过关联小文件及元数据的预取方案，有效地提高了 I/O 性能，解决了 HDFS 上存储小文件存在的问题。对 50000 个小文件进行实验，结果表明通过小文件合并及相关缓存预取策略使内存占用相对原始 HDFS 系统减少了 19%，读取效率相对原始 HDFS 提高了 20%。接下来将进一步改善解决方案中的合并策略，进一步研究小文件之间的联系从而尽可能地将属于同一类课程的小文件合并到一个大文件中，将关联关系最强的小文件放在相邻存储空间，以更好地实现关联小文件的预取策略，提高小文件预取准确率，进而更好地改善小文件在 HDFS 上的读取效率。

参 考 文 献

[1] kkdelta. 告诉你 Hadoop 是什么[EB/OL]. [2014-06-17]. http://www.thebigdata.cn/Hadoop/10722.html

[2] 周敏奇，王晓玲，金澈清，等. Hadoop 权威指南(第 2 版)[M]. 北京：清华大学出版社，2011：8-20

[3] White T. The small files problem [EB/OL]. [2009-2-2]. http://www.cloudera.com/blog/2009/02/the-small-files-problem

[4] Dong Bo, Qiu Jie, Zheng Qing-hua, et al. A novel approach to improving the efficiency of storing and accessing small files on Hadoop: a case study by powerpoint files [C]// IEEE International Conference on Services Computing. Miami, Florida, Piscataway: IEEE, 2010: 65-72

[5] 李宽. 基于 HDFS 的分布式 Namenode 节点模型的研究 [D]. 广州：华南理工大学，2011
Li Kuan. Research of the Model of Distributed Namodes in HDFS[D]. Guangzhou: South China University of Technology, 2011

[6] 赵晓永，杨扬，孙莉莉，等. 基于 Hadoop 的海量 MP3 文件存储架构[J]. 计算机应用技术，2012，32(6)：1724-1726
Zhao Xiao-yong, Yang Yang, Sun Li-li, et al. Hadoop-based storage architecture for mass MP3 files[J]. Journal of Computer Applications, 2012, 32(6): 1742-1726

[7] Fu Song-ling, Huang Chen-lin, He Li-gang, et al. iFlatLFS: Performance Optimization for Accessing Massive Small Files[C]// 20th International Conference on High Performance Computing. Bangalor, Piscataway: IEEE, 2013: 10-19

[8] Li Jia, Lin Kun-hui, Wang Jing-jin. Design of the Mass Multimedia Files Storage Architecture Based on Hadoop[C]// the 8th International Conference on Computer Science & Education. Colomlo, Piscataway: IEEE, 2013: 801-804

[9] 王涛，姚世红，徐正全，等. 云存储中面向访问任务的小文件合并与预取策略[J]. 武汉大学学报(信息科学版)，2013，38(12)：1504-1508
Wang Tao, Yao Shi-hong, Yu Zheng-quan, et al. A Small File Merging and Prefetching Strategy Based on Access Task in Cloud Storage[J]. Geomatics and Information Science of Wuhan University, 2013, 38(12): 1504-1508

[10] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, et al. A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System[C]// 2013 International Conference on Computer Communication and Informatics (ICCCI). Coimbatore, Piscataway: IEEE, 2013: 1-8

[11] 郑庆华，董博，刘均，等. 一种基于 Hadoop 的海量可归类小文件关联存储方法：中国，102332029A[P]. 2012-01-25

(上接第 49 页)

[5] Platform specific meta-model of deployment environment of Google App Engine[OL]. http://download.csdn.net/download/u012261044/8386821

[6] Bunch C, Drawert B, Chohan N, et al. Language and Runtime Support for Automatic Configuration and Deployment of Scientific Computing Software over Cloud Fabrics[J]. Journal of Grid Computing, 2012, 10(1): 23-46

[7] Brandtzæg E, Mohagheghi P, Mosser S. Towards a Domain-Specific Language to Deploy Applications in the Clouds[C]// CLOUD COMPUTING 2012: The Third International Conference on Cloud Computing, GRIDs, and Virtualization. 2012

[8] Wettinger J, Andrikopoulos V, Strauch S, et al. Enabling Dynamic Deployment of Cloud Applications Using a Modular and Extensible PaaS Environment[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing. 2013: 478-485

[9] D'Andria F, Bocconi S, Cruz J, et al. Cloud4SOA: Multi-cloud Application Management Across PaaS Offerings[C]// Proceedings of the 2012 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'12). 2012: 407-414

[10] Quinton C, Haderer N, Rouvoy R, et al. Towards multi-cloud configurations using feature models and ontologies[C]// Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds (MultiCloud'13). 2013: 21-26

[11] Ferry N, Rossini A, Chauvel F, et al. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD'13). 2013: 887-894

[12] Nguyen B M, Tran V, Hluchy L. Development and deployment of cloud services via abstraction layer[C]// 2013 International Conference on Computing, Management and Telecommunications (ComManTel). 2013

[13] Sellami M, Sami Yangui, Mohamed M, et al. PaaS-Independent Provisioning and Management of Applications in the Cloud[C]// Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing (CLOUD'13). 2013: 693-700

[14] Ma Zhi-yi, He Xiao. Building Modeling Tools Based on Metamodeling and Product Line Technologies[J]. Chinese Journal of Electronics, 2014, 23(2): 219-226