一种基于执行力模型的服务平台自主控制方法

顾 军[1),2)] 罗军舟[1)] 曹玖新[1)] 李 伟[1)]

[1)](东南大学计算机科学与工程学院 南京 211189)

[2)](中国矿业大学计算机科学与技术学院 江苏徐州 221116)

摘 要 开放的分布式服务平台倾向于涵盖更丰富的管理功能,支持更强的分散交互性,从而导致软件管理和维护的难度和成本问题日益突出.为此,引入一种自管理的服务平台体系结构参考模型,以构件作为功能实现载体,服务作为功能组织手段,交互作为功能扩展方式.提出了一种基于分层反馈的自主控制架构,以服务构件及相互之间的交互关系为控制对象,执行力模型为决策基础.在管理服务的可用性和性能建模中运用了马尔可夫过程、随机Petri网和排队网模型理论,并考虑了链路和节点的失效修复机制.仿真结果表明,基于排队Petri网的执行力模型能够反映失效率和修复时间对服务平台性能和可用性的影响,并验证了自主控制方法对提高服务平台有效性的积极作用.

关键词 服务平台;自主控制;执行力;服务构件;交互;排队Petri网

中图法分类号 TP393 **DOI号**: 10.3724/SP.J.1016.2012.00282

Service Platform Autonomic Control Approach Based on Performability Model

GU Jun[1),2)] LUO Jun-Zhou[1)] CAO Jiu-Xin[1)] LI Wei[1)]

[1)](*School of Computer Science & Engineering, Southeast University, Nanjing* 211189)

[2)](*School of Computer Science & Technology, China University of Mining and Technology, Xuzhou, Jiangsu* 221116)

Abstract The open and distributed service platform is prone to containing more management functions and supporting stronger decentralized interaction, so the hard-management and high-cost problems become more serious. For that, a self-managing service platform architectural reference model, characterized by component-based, service-oriented and interaction-centric, is introduced in this paper. On this ground, a hierarchical autonomic control structure based on feedback mechanism is proposed, which takes the service components and their interaction relations as controlled objects, and makes decision according to the analytical performability model of service platform. A combination of availability and performance models of management service is suggested via Markov Procedure, Stochastic Petri Net and Queueing Network theory. Moreover, the ability of failure recovery into the service components and links are taken into account. Numerical examples show that the proposed performability model based on Queueing Petri Net can quantitatively analyze the influence of failure probability and recovery time on availability and performance of service platform, and autonomic control approach is helpful for improving the effectiveness of service platform.

收稿日期:2009-12-25;最终修改稿收到日期:2010-09-06.本课题得到国家"九七三"重点基础研究发展规划项目基金(2010CB328104)、国家自然科学基金(60903161,60903162,61003257,61070161,61070158,61003311),高校博士点专项基金(200802860031,20110092130002)、江苏省自然科学基金重点项目(BK2008030)、江苏省网络与信息安全重点实验室(BM2003201)和计算机网络与信息集成教育部重点实验室(93K-9)资助.**顾 军**,男,1977年生,博士研究生,讲师,主要研究方向为网络与服务计算.E-mail: jgu@seu.edu.cn.**罗军舟**,男,1960年生,博士,教授,博士生导师,主要研究领域为下一代网络体系结构、协议工程、网络安全、网格与云计算、无线局域网.**曹玖新**,男,1967年生,博士,教授,博士生导师,主要研究领域为网络安全、移动网络与服务计算、面向复杂网络的信息传播行为.**李 伟**,男,1978年生,博士,副教授,主要研究方向为下一代互联网、网络管理和服务计算.

Keywords　service platform; autonomic control; performability; service component; interaction; queueing Petri net

1　引　言

在面向服务的分布式系统架构中，服务以可共享与可集成的资源为基础来构建，通常表现为一个独立的业务功能. 相比于对象、构件等早期形式，服务具有更大的粒度、更强的独立性和更松散的耦合性，从而为形成灵活、动态的服务组合带来了方便[1]. 由全球网格论坛 GGF 提出的开放网格服务架构 OGSA(Open Grid Services Architecture)[2] 采用服务作为资源封装和互操作的统一形式，通过定义基于 OGSI/WSRF 的标准框架协议实现面向服务的开放网格系统. 在这样的系统中，高层商业目标、内部组织规模和外部运行环境均会不断变化，因此客观上要求面向服务的系统平台具有动态适应能力，能够感知环境和用户需求的变化，并根据这些变化调整自身的结构和行为，以实现特定的目标，包括资源的动态配置、服务的动态合成、系统参数的动态校正、软硬件故障的动态修复等. 如果单靠 IT 专家和技术人员依据系统状态和高层的管理策略直接对系统进行干预和处理，不但反应慢、效率低，而且成本也高[3]. 如何使服务平台能够根据管理者的要求实现自我管理已经成为亟待解决的重要问题之一[4].

在本文的前期工作中，面向服务的网格平台被看作是应用管理、执行管理、安全管理、资源管理、数据管理、信息服务等多种管理功能软件实体的集合，它们之间通过分布式的交互和协作共同完成全局的管理任务和目标[5]. 所有管理功能都设计成服务构件(service component)形式，以构件作为功能实现载体，以服务作为功能组织手段，随同实现服务的构件一起部署. 服务构件可以是多种子服务构件的组合体，支持需求驱动的服务构件的选择、匹配、组装和协同，从而为管理者和使用者提供高效、灵活的管理服务定制功能. 这样的设计提升了服务平台的可管理性，包括状态可观测和状态可调整，前者支持监测度量，后者支持控制执行. 然而，究竟采用什么方法对服务平台的自管理行为进行控制需要进一步研究.

文献[3]将现有的自管理技术方法归纳为两种基本类型：知识模型方法和数学模型方法. 知识模型方法运用人工智能或知识工程的方法和技术所建立的知识模型，在知识库中建立状态判定、策略和问题求解等 3 类知识，通过基于知识分析和逻辑推理进行决策. 该方法只能进行定性分析和逻辑推理，不适合那些需要定量描述系统的有关过程和特性的场合，如对系统响应速度、吞吐量、CPU 和存储器的利用率等系统性能的管理. 数学模型方法一般运用控制论或运筹学的理论和方法来建立，能够依据不断改变的资源和环境状态自主决定系统参数的调整，使得系统性能保持在期望的范围内. 对于网格计算和云计算这种大规模、跨多管理域的虚拟计算环境，资源故障和构件失效的情况相当普遍，需要综合考虑系统的可用性和性能，即执行力(performability)[6]. 执行力可以看作是对系统在给定时间间隔内获得和维护满意的性能级别能力的量化，在此过程中需要考虑系统结构和环境的改变. 以执行力模型作为自主控制的决策基础，有助于把握服务平台的内部结构和外部环境的不确定特性，并保持服务平台运行的高性能.

因此，本文以面向 Internet 的开放、动态、难控的网络环境为基本驱动力，以实现服务平台软件维护和管理的自主性为目标，研究一种基于执行力模型的服务平台自主控制方法. 该方法能够有效降低服务平台中间件的复杂性，主要体现在以下三方面：(1) 软件实体的服务化和构件化提升了中间件的轻量化水平；(2) 自主控制实现框架的建立使得中间件能够按照管理需求和运行环境的变化进行动态调整和演化，使系统具有尽可能高的用户满意度；(3) 综合考虑可用性和性能的执行力模型有利于全面预测和评估中间件的执行状态，提高触发自主控制机制实施的准确度和时效性.

本文第 2 节介绍相关工作；第 3 节提出一种自管理的服务平台体系结构参考模型，介绍服务构件的交互关系；第 4 节阐述基于执行力模型的服务平台自主控制方法，介绍基于分层反馈控制的实现框架，重点研究服务构件和管理服务的执行力建模方法；第 5 节给出实验和数据分析；第 6 节总结全文并展望下一步的工作.

2　相关工作

自主计算由 IBM 公司在 2001 年发起[7]，旨在

参照自主神经系统的自我调节机制,以现有理论和技术为基础构建自主计算系统,使得信息系统整体上实现自我管理. Kephart 等人[8]于 2003 年提出 MAPE(监视-分析-规划-执行)控制环机制,在 2004 年提出由动作策略、目标策略和效用函数策略组成的统一框架[9];Kephart[10]还以效用函数策略作为反映高层目标的手段,建立了一种用于原型数据中心的资源自主管理和分配模型. Hinchey 等人[11]在自主计算的基础上提出了自我管理软件的概念,提供了一个软件开发和演化的整体观,有望把系统的自动化、自治和可靠性带到一个新的水平.

自主计算思想已经在学术和工业界得到了广泛应用. 文献[12]在面向网格的 ASSIST 编程环境中实现了一个具有 QoS 控制能力的自主应用管理者原型;文献[13]提出通过对网格构件模型(Grid Component Model)的行为骨架(Behavioural Skeletons)实现网格构件的自主管理;AutoMate 项目针对网格环境中存在的复杂性、动态性、异构性和不确定性等挑战,给出了一种自主系统的概念模型和实现体系,开发了一个能够根据高级任务的目标和约束动态使用基本网格服务的自主组合引擎[14-15];AutoGrid 项目[16]致力于在 Integrade 网格基础设施之上增加自管理能力,减少人对系统配置和管理的干涉;文献[17]将构件模型与结构化覆盖网(structured overlay networks)相结合,提供大规模分布式应用的自管理能力;文献[18]提出网格单元(grid unit)概念,通过代理技术构建自管理框架. OptimalGrid 项目研究如何简化大规模平行网格应用的创建和管理[19]. 文献[20]提出把自主计算、网格计算和虚拟化技术结合起来,建立自主的商务网格的思想. 这些研究虽然提高了网格系统面向应用的自管理能力,但并没有考虑网格中间件本身的可管理性和可重用性问题,易导致更高的系统复杂性.

研究人员在系统的可运行性建模和分析方面已经开展了许多工作[21]. 分析性数学模型可分为确定模型和随机模型等[22]. 排队模型是一种确定性的数学模型,是很好的网络建模工具. 随机模型是以随机过程为基础,主要包括马尔可夫回报模型、随机进程代数以及随机 Petri 网等,这些基于状态的随机方法更容易对系统状态进行全面有效的描述,精确刻画系统随机行为以及组成部件之间的相互关系,便于计算各种分析指标. Ranami 和 Trivedi 等人[23-24]经过对 CORBA 事件服务和通知服务的性能分析,给出了分布消息服务的一种可运行性建模框架[25],进一步研究了对实时系统响应时间分布建模的技术[26]. 刘显明等人[27]考虑了网格环境和监控信息的特点,使用随机 Petri 网建立网格监控体系结构的可运行性模型,并讨论了系统可用性、响应时间分布、事件丢失概率、公平性等问题. Das 等人[28-29]使用分层排队网对多层服务系统的可运行性进行了建模,并给出了基于马尔可夫链的分析方法. Qu 等人[30]采用随机回报网(Stochastic Reward Nets, SRN)对计算网格的分层资源调度算法的可运行性进行了评价. 基于 SOA 的分布式系统采用松耦合和重组装的开发模式,随着组合服务的规模增大,基于随机回报网的可运行性模型越来越复杂. Kogekar 等人[31]针对这个问题,提出了一种模型驱动的通用框架,实现多个 SRN 模型自动化综合和仿真. Bernardi 等人[32]对基于概率模型的可信赖评价方法进行了研究,并给出了利用随机 Petri 对复杂系统进行建模和评价的方法. 林闯等人[33]研究了随机 Petri 网对网络系统可信赖性建模分析的方法和步骤,着重研究了随机 Petri 网描述系统的服务失效模型和容错模型,并给出了网络可信赖性分析中主要指标的计算方法. 为了更加有效地对管理服务特性进行定量分析,特别是对动态性的描述,需要在上述研究的基础上建立相应的数学模型用于定量描述服务构件的有关动态和静态过程. 服务平台的管理行为具有随机性,并满足马尔可夫特性,可以用排队论建立性能模型. 排队 Petri 网(Queueing Petri Nets, QPN)[34-35]继承和发展了排队网和随机 Petri 网的优点,可用于描述服务管理的动态行为、性能指标和可用性,因此我们采用分层排队 Petri 网对服务平台执行力进行建模和评价.

3 自管理的服务平台参考模型

分布式服务平台不仅要管理网络、设备、数据、任务、用户等多种对象,还要综合考虑 QoS、安全、性能、可靠、可信等指标对管理行为的约束,所以其软件构成将会很丰富,实现起来也很复杂. 特别是随着服务平台开发、部署、运行和维护的外部环境逐渐从封闭、静态、可控走向开放、动态、难控,不能再假设平台中各个部分都遵从统一的设计和管理,也不能完全精确地预先确定软件的结构组成和各个部分的行为,更无法界定软件部件与外部环境的边界,因此服务平台软件的网络化和构件化成为新的发展趋势[36]. 首先定义相关概念如下.

定义 1.　管理功能集合 $MF_{\text{set}}=\bigcup_{i=1}^{n} mf_i$，其中 $mf_1 \cap mf_2 \cdots mf_{n-1} \cap mf_n=0$.

定义 2.　管理功能子集 $MF_{\text{subset}}=\prod_{i=k}^{l} mf_i$，即管理功能子集是功能集合 MF_{set} 的一个子集合，同时这些子功能之间又有一定的逻辑关系，而不是简单的联合.

定义 3.　管理服务 $MS_{\text{set}}=\bigcup_{j=1}^{m} MS_i$，即所有管理子服务的集合. 一个管理子服务 MS_i 总是与一个管理功能子集相对应，是管理功能的抽象表示，也可看作是面向应用的管理功能的逻辑实现.

定义 4.　服务资源集合 $SR_{\text{set}}=\bigcup_{j=1}^{m} SR_i$，即所有支持服务执行的资源总和，包括虚拟资源和物理资源.

定义 5.　服务构件子集 $SC_{\text{set}}=\prod_{j=k}^{l} SC_j$，即服务构件子集是服务资源集合 SC_{set} 的一个子集合，与功能子集一样，这些子服务构件之间有一定的组合和协作关系.

本文提出的自管理的服务平台体系结构参考模型是一个用以指导建立具体分布式应用系统的抽象框架(如图 1 所示)，描述了平台中间件的环境设施以及其中的部件和部件之间的关系，但不直接受任何具体标准、技术或其它实现细节的约束.

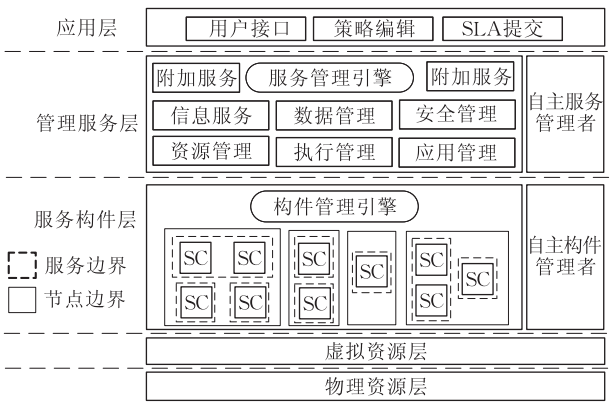

图 1　自管理的分布式服务平台体系结构

参考模型自下而上分为 5 层：

(1) 物理资源层(physical resource layer)是实际的软硬件资源集合.

(2) 虚拟资源层(virtual resource layer)是对物理资源的服务化封装和描述.

(3) 服务构件层(service component layer)是完成特定管理功能的服务构件集合，提供实现服务所需要的各种环境和构件，负责用细粒度的构件实现粗粒度的服务. 一个服务构件(SC)实现一种管理功能，通常没有明确的业务目标. 服务构件的粒度有大小之分，功能有强弱之判，层次有高低之别. 由于环境的动态性和管理的分散性，服务构件需要在不同的时刻实施不同的协同行为. 若干服务构件可以在构件管理引擎(component management engine)的作用下协作完成更大的功能. 服务构件组合(service component composite)以上层的管理服务目标为操作指导，可以实现跨节点的构件交互，是保证在不同抽象层次上对中间件进行描述、分析和验证的基础，也是实现管理服务定制的基础. 自主构件管理者(autonomic component manager)负责对执行状态的服务构件提供自管理机制，能够在运行过程中对外部环境和应用需求的变化做出适当反应，从而将系统提供的服务的功能或性能维持在一个令人满意的水平上.

(4) 管理服务层(management service layer)包含了面向应用的各种管理功能的服务实例. 管理服务的功能设计不是固定的，可以根据管理需求动态扩展，如附加服务(pluggable service). 服务管理引擎(service management engine)负责服务发现、服务组合、服务协同等工作. 自主服务管理者(autonomic service manager)负责实现管理服务的自管理，包括服务的监控、分析、规划和执行等.

(5) 应用层(application layer)对外提供与用户交互的接口，获取用户的管理需求(如计算密集型、数据密集型、业务流程型等)和服务质量要求，制定或调整管理功能的部署规划和执行策略.

为适应大规模网络环境而采取的分布式管理模式使得服务构件的数量快速扩张，增加了管理和维护的难度和成本. 研究表明，引起系统复杂行为的主要原因不是元件的数量而是元件之间的交互，只要能保持系统元件之间交互的基本性质，那么即使对系统加以简化，系统的基本特性也不会改变[37]. 因此，本文提出的自管理服务平台采用基于交互研究服务构件的组织模式. 不同的交互关系和交互方法将会对整体的管理服务的性能、可靠性、可信性等产生重大影响. 在管理需求和策略指导下，分散在不同节点上的服务构件可能存在的交互关系[38]如下.

定义 6.　假设有服务构件 SC_i 和 SC_j，如果 SC_i 执行完毕后，SC_j 接着开始执行，那么称之为顺序方式(sequence style)，记作 $SC_i;SC_j$.

定义 7. 假设有服务构件 SC_i，如果 SC_i 重复执行若干次，那么称之为重复方式(iterative style)，记作 μSC_i.

定义 8. 假设有服务构件 SC_i 和 SC_j，如果 SC_i 和 SC_j 相互独立执行，那么称之为并发方式(concurrent style)，记作 $SC_i \| SC_j$.

定义 9. 假设有服务构件 SC_i 和 SC_j，如果 SC_i 和 SC_j 并行执行以提供相同的管理服务功能，那么称之为容错方式(fault-tolerant style)，记作 $SC_i | SC_j$.

从可用性和性能角度考察服务构件的交互关系，sequence 和 concurrency 两种方式下服务构件的执行力相互制约，iterative 方式可以看作 k 个服务构件按照 sequence 方式连续工作，而 fault-tolerant 方式的服务构件独立工作、择优选用，相互之间没有直接影响.

4 基于执行力模型的自主控制方法

4.1 分层的反馈控制架构

服务平台自主控制方法以反馈控制为基础，主要的系统构造任务包括：分析服务平台体系结构并把它建模为一个反馈控制系统；以服务构件、管理服务以及相互之间的交互关系作为控制对象；通过对控制对象的描述和建模，把执行力控制问题映射为一个系统的控制循环；选择适当的传感器来动态监控当前系统运行状态，及时预测和定位可能存在的失效或缺陷环节；指导效应器根据一定的策略和规则实现系统行为的自适应调整.

本文提出的基于分层反馈的自主控制实现架构如图 2 所示，下层负责面向服务构件的局部控制，上层控制面向应用的全局管理服务. 全局控制器所使用的系统模型是抽象模型，它包含与全局目标相关的信息，如用于刻画服务构件之间的交互细节的局部变量. 全局控制器管理服务平台的宏观特性，它通过命令的方式对每个局部控制器施加操作约束. 每个局部控制器则基于操作约束优化本地的管理服务性能.

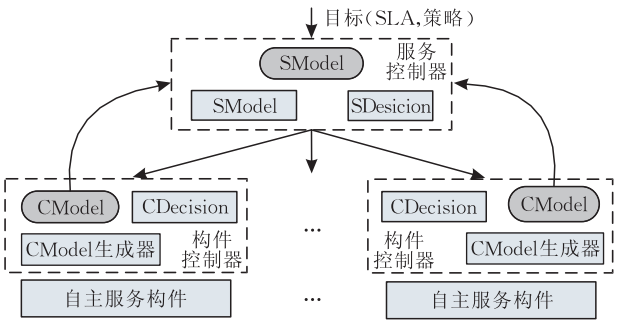

图 2 基于分层反馈的自主控制架构

构件控制器(component controller)表示为三元组：$CC=(CM,CMB,CD)$. 式中，CM 是构件的分析性模型(CModel)，负责建模结果的表示和存放；CMB 是模型生成器(CModel Builder)，负责定量描述系统的有关过程和特性，如对系统响应速度、吞吐量、CPU 和存储器的利用率、失效率等性能度量的管理，对系统可靠性、可用性和安全性的评估等；CD 是决策部件(CDecision)，实现服务构件状态偏离期望目标时的自适应功能. 服务控制器(service controller)的组成与构件控制器基本一致，可以表示为三元组：$SCon=(SM,SMB,SD)$. 不同的是服务的分析性模型(SModel)是由下层服务构件的分析模型通过组合计算得到，反映了全局的执行状态. 服务控制器部件在一定的外部指导支持下运行，常见的外部指导有：SLAs 是用户提交的 QoS 需求，策略(policy)是表征高层管理目标的任何形式化规范，用于驱动和约束构件和服务的行为.

自主服务构件(autonomic service component)是自主控制对象，具有监控、决策和控制 3 种功能，可以实现自我觉察和外部环境感知，其形式化模型见文献[5]. 自主服务构件的种类和数目取决于实际应用对服务平台管理功能的需求类型和应用请求的规模，反映在软件开发环节上就是不同功能的软件实体的参数接口、内部逻辑和调用关系不一样. 但是从软件维护与管理的角度，可以把分布式中间件平台软件抽象为一组分布于 Internet 环境下各个节点的、具有主体化特征的软件实体，以及一组用于支撑这些软件实体以各种交互方式进行协同的连接子，即服务构件. 因此，本文只考虑服务构件对外表现的管理功能的差异，不考虑服务构件内部的实现细节. 由于服务平台管理工作是开放、动态和应用驱动的，服务构件的组织不拘束于某个固定形式，应用需求、管理目标和策略的不同决定了服务构件组成和执行序列的差异. 对这些服务构件的管理主要体现在服务构件负责哪些管理功能，服务构件在节点上如何分布，服务构件之间的交互关系是怎样的，即通过不同管理方案的设计，选择服务构件的数量、实现流程和路径，对不同的组合方案的性能和可靠性进行预测和评估，从而为不同的对象管理目标选择一个较优的管理方案.

服务平台自主控制的目标可以定义为当预测到

管理服务在执行过程中可能违背期望的约束指标时，服务控制器(SCon)选择合理的策略调整管理服务中服务构件的工作和组织方式.为此，首先要明确自主控制对象，给出关于管理服务的类型、强度、服务质量等的描述，定义需要的服务构件的类型、数量、性能参数、可用性指标、可靠性度量、执行依赖关系序列等，形成方案文件；然后建立控制对象的执行力模型，通过模型的仿真数据分析可以预先判断局部服务构件和整体管理服务的特点，如平均服务时间、可用性程度等；最后参照分析结果和决策依据做出判断，完成下一步工作.

4.2　执行力度量

根据分布式计算环境的动态性和虚拟化特征，服务平台执行力取决于资源执行力、构件执行力和服务执行力[39].资源执行力是对构件运行环境的抽象和量化.资源的虚拟化使得资源执行力的测量变得更加复杂，需要解决资源的动态性和非专注性的问题.构件执行力描述构件自身的性能属性和可用性度量，构件开发者需要将构件的这些属性和度量加入到构件的描述中.构件的性能需要和实际的运行环境兼容，在特定环境下进行实例化，并监控构件的可靠性和有效性.服务执行力针对特定应用域中系统完成某次行为所需构件以及构件之间的交互，它可以从系统设计的用例图和时序图中得到.构件之间通过接口进行交互，会触发多种执行行为的发生.由此可见，资源执行力对外表现为构件功能的运行，服务执行力对外表现为构件行为的变迁.

服务构件可能同时服务于多个管理服务域，同一个服务构件在不同的运行过程中可能产生各种反映服务构件当前状态的数据(事件)，包括各种日志事件和实时事件，如服务的操作状态、性能状态、失效征兆和异常事件等[40].讨论之前，假定所有服务构件都是第三方构件，它们的细节处理被排除在外，并且服务构件在相应的资源上能够运行.

执行力度量的最普通形式就是系统各属性的加权和，每个属性表征了对象的特性.执行力函数的一种格式为

$$f_{PerS}=\sum_{i=1}^{n}V_i(x_i) \tag{1}$$

其中：V 是属性 i 的相对重要性的加权条件变量；x_i 是属性 i 的值.通常，对归一化的 f_{PerS} 来说，$\sum_{i=1}^{n}V_i=1$.

但是，要确定不同属性的权值往往非常困难，而且不同属性在不同交互关系下的计算方式也是不同的.例如在顺序方式下，响应时间(response time)和延迟(delay)具有叠加性，失效概率(failure probability)和修复概率(recovery probability)具有连乘性，失效速率(failure rate)和修复速率(recovery rate)是独立分布的.因此很难直接用来计算服务构件执行力.此外，在实际的服务平台中，管理服务的处理能力往往受到服务构件数量、服务请求规模、管理强度、资源能力和外来干扰等的影响，参数改变具有一定的随机性和不确定性，需要动态修正计算结果，增加了计算复杂性.

因此，本文认为服务平台执行力模型是从用户的角度反映服务构件完成管理任务的执行能力，一般由可用性模型、性能模型和结合方法三部分组成.其中，可用性模型描述了管理任务被服务平台成功完成的概率，不但考虑到服务构件软硬件和连接环节的失效对可靠性的影响，而且关注失效可修复情况下系统服务可用性的统一表达.性能模型描述了不同的管理服务构件执行序列的执行效率，需要考虑响应时间、延迟、吞吐量、丢失率等属性.可用性模型与性能模型的结合主要表现为引入失效修复机制后对性能属性参数的影响.

4.3　可用性模型

网络环境的开放、动态和分布性特征导致资源故障和构件失效的情况相当普遍.为了更好地描述失效对服务平台可用性的影响，对节点、链路以及服务构件软件的失效情况做如下的基本假设：(1)链路是节点之间的虚拟链路，不考虑具体的链路拓扑结构；(2)服务平台中节点、链路和各服务构件之间相互独立，它们的失效是统计独立的；(3)运行服务构件的节点在获得管理任务后立即执行，并且其无失效执行时间服从指数分布；(4)服务构件之间在交换信息期间，链路无失效工作的时间服从指数分布；(5)需要较长时间和占用较多计算机资源的“大任务”，其节点可靠性随着时间的增加而呈指数衰减[41].

令管理服务的总复杂度为 C，由 m 个相互独立的管理服务构件组成，其中第 i 个服务构件 SC_i($i=1,2,\cdots,m$)的复杂度为 c_i.由于服务构件之间的独立性，C 可以表示为各服务构件的复杂度之和，即

$$\sum_{i=1}^{m}c_i=C \tag{2}$$

服务平台根据网络资源的分布情况，将服务构件 SC_i 分配到节点 N_k 来执行.如果节点 N_k 的 CPU 处理速度为 sp_k，那么在不发生失效的情况下，服务构

件 S_i 在节点 N_k 的执行时间为

$$\tau_{ki} = \frac{c_i}{sp_k} \tag{3}$$

软件出现错误是不可避免的，由软件错误导致的软件失效也是不可避免的[42]. 为了更好地描述软件可靠性对服务构件的影响，假设执行服务构件 SC_i 所调用程序的失效率为 $\lambda_p(i)$，运行服务构件的节点的失效率为 $\lambda_n(k)$，那么服务构件执行时不发生失效的概率为

$$R_{ki}(i) = e^{-\lambda_p(i)\cdot\tau_{ki}} e^{-\lambda_n(k)\cdot\tau_{ki}} = e^{-\frac{\lambda_p(i)+\lambda_n(k)}{sp_k}c_i} \tag{4}$$

网络通信环境和服务构件的交互方式是服务构件行为变迁必须要考虑的影响因素. 假设通信链路带宽为 B，链路失效率为 λ_L，服务构件 SC 交互的总次数为 K，按照 C/S 方式调用构件后第 i 次传输的数据为 D_i，按照移动代理方式调用构件需要传输的数据为 D_r，移动代理迁移过程中需要传输的数据为 D_m. 那么，服务构件之间通信时不发生失效的概率为

C/S 方式下

$$R_{L(C/S)} = \prod_{1\le i\le K} e^{-\frac{\lambda_N\cdot D_i}{B}} \tag{5}$$

移动代理方式下

$$R_{L(Ag)} = e^{-\frac{\lambda_N\cdot(D_m-D_r)}{B}} \tag{6}$$

基于上述分析，加之节点失效和链路失效具有独立性，那么服务构件 SC_i 能成功地通过网络获取数据并被节点 N_k 执行完成的概率为

$$R_{i,(k)} = R_{ki}(i)R_L(i) \tag{7}$$

在实际的服务平台中，服务构件 SC_i 的执行往往因为所在节点 N_k 发生失效而被迫终止. 为此，引入节点失效修复机制，即当节点发生用户误操作、CPU 资源短时衰竭、网络瞬时堵塞或短期中断等可修复故障时，节点自动运行失效恢复程序，修复已停止的服务构件执行程序，这样就可以有效解决管理服务遇到失效而终止的问题. 当然，失效修复具有一定的概率，即失效可以修复，修复后节点 N_k 从失效断点继续执行管理服务构件，直到遇到不可修复的失效或者服务构件 SC_i 执行完毕. 反之，失效不可修复，SC_i 被迫终止，在该节点的管理任务执行宣告失败. 如果运行服务构件的节点之间只有一条通信链路，那么在服务构件交换信息期间通信链路发生了失效，服务构件的执行将失败，管理任务也随之被终止.

服务构件的失效行为可以通过连续时间马尔可夫链(Continuous Time Markov Chain, CTMC)加以描述[25]，如图 3(a)所示. 其中，UP 代表服务构件运行正常，PF 代表服务构件程序失效，失效率为 λ_p，修复率为 γ_p，NF 代表服务构件节点失效，失效率为 λ_n，修复率为 γ_n，成功修复的概率为常数 cp.

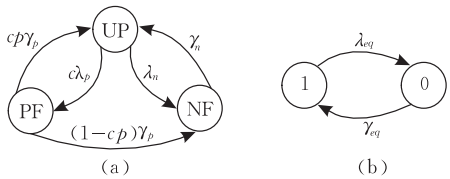

图 3 失效行为的 CTMC 模型

因此，服务构件 SC_i 的稳定状态概率计算如下：

$$\begin{cases} \pi_{UP} = \left[1+\dfrac{\lambda_p}{\gamma_p}+\dfrac{1}{\gamma_n}(\lambda_n+(1-cp)\lambda_p)\right]^{-1}, \\ \pi_{CF} = \dfrac{\lambda_p}{\gamma_p}\pi_{UP}, \\ \pi_{NF} = \dfrac{1}{\gamma_n}(\lambda_n+(1-cp)\lambda_n)\pi_{UP} \end{cases} \tag{8}$$

服务构件 SC_i 可用性为 $A_{SC_i}=\pi_{UP}$.

图 3(a)的两状态等价可用性模型如图 3(b)所示，1 表示服务构件运行正常(up)，0 表示服务构件运行失效(down). 令等价的失效率和修复率分别表示为 λ_{eq} 和 γ_{eq}，那么可用性表示为

$$\begin{cases} \lambda_{eq} = \dfrac{\pi_{UP}(\lambda_p+\lambda_n)}{\pi_{UP}} = \lambda_p+\lambda_n, \\ \gamma_{eq} = \dfrac{\pi_{CF}cp\gamma_p+\pi_{NF}\gamma_n}{\pi_{CF}+\pi_{NF}}, \\ A_{SC_i} = \dfrac{\gamma_{eq}}{\lambda_{eq}+\gamma_{eq}} \end{cases} \tag{9}$$

不同交互方式下服务构件的可用性描述如下：

① 顺序方式

服务平台中的服务构件 SC_1 和 SC_2 之间通过链路 L 通信实现交互与协作，那么它们的可用性模型可表示为一个 8 状态的连续时间马尔可夫链，如图 4 所示.

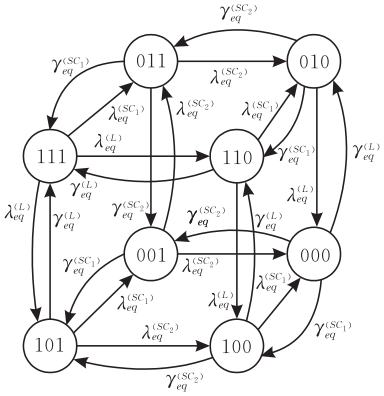

图 4 8 状态 CTMC 的可用性模型

分别以上标 (SC_1)，(SC_2) 和 (L) 标识交互的两个服务构件和链路的失效率和修复率，每个状态表示

一个三元组(SC_1,L,SC_2),那么顺序执行方式下的可用性为 $A_{SC_1;SC_2}$(即所有状态的稳定状态概率)见表 1.

表 1　8 状态 CTMC 的稳定状态概率

状态	稳定状态概率
111	$A_{SC_1}A_LA_{SC_2}$
011	$(1-A_{SC_1})A_LA_{SC_2}$
101	$A_{SC_1}(1-A_L)A_{SC_2}$
110	$A_{SC_1}A_L(1-A_{SC_2})$
001	$(1-A_{SC_1})(1-A_L)A_{SC_2}$
010	$(1-A_{SC_1})A_L(1-A_{SC_2})$
100	$A_{SC_1}(1-A_L)(1-A_{SC_2})$
000	$(1-A_{SC_1})(1-A_L)(1-A_{SC_2})$

② 重复方式

假设服务构件 SC_1 重复运行 μ 次后进入下一个服务构件执行状态,那么循环方式下的可用性 $A_{\mu SC_1}$ 为

$$A_{\mu SC_1}=(A_{SC_1})^{\mu} \tag{10}$$

③ 并发方式

假设 SC_1 和 SC_2 各自独立运行后进入同一个服务构件的执行状态,它们的可用性模型可表示为一个 4 状态的连续时间马尔可夫链,如图 5 所示.

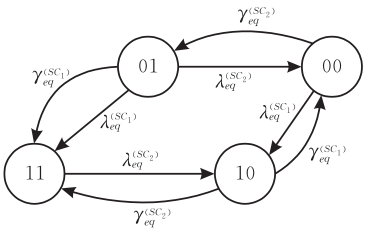

图 5　4 状态 CTMC 的可用性模型

那么并发方式下的可用性 $A_{SC1\parallel SC2}$(即所有状态的稳定状态概率)见表 2.

表 2　4 状态 CTMC 的稳定状态概率

状态	稳定状态概率
11	$A_{SC_1}A_{SC_2}$
01	$(1-A_{SC_1})A_{SC_2}$
10	$A_{SC_1}(1-A_{SC_2})$
00	$(1-A_{SC_1})(1-A_{SC_2})$

④ 容错方式

假设 SC_1 和 SC_2 同时运行提供相同的服务管理功能,其中一个完成后进入下一个服务构件的执行状态,那么容错方式下的可用性 $A_{SC_1|SC_2}$ 为

$$A_{SC_1|SC_2}=1-(1-A_{SC_1})(1-A_{SC_2}) \tag{11}$$

在上述研究的基础上,只要能够明确管理服务中的服务构件数量、执行序列图和相关参数,就可以从理论上对基于服务构件的分布式管理服务的可用性进行分析和计算.

4.4　执行力模型

4.4.1　建模服务构件执行力

服务平台中的服务构件具有分布性,因此通信环节是必须要考虑的环节.本文将服务构件看作是链路模型和节点模型的结合,并满足以下假设:(1)所有失效都是可修复的;(2)链路和节点的失效是统计独立的.由于网络资源的共享性,管理节点上往往运行着除管理服务外的其它任务,这些额外的负载也会影响管理服务的响应时间.因此,每个服务构件包括 4 类队列,分别建模通信链路(LP)、修复环节(LR,NR),服务节点(NP)和外部负载(EW).每类队列刻画了待处理对象的序列关系和性能特征.在可用性建模中引入节点失效修复机制后,服务构件的生命周期分为有效执行和失效修复 2 个阶段.以通信链路为例,在有效执行阶段,通信任务持续进入队列 LP,完成后进入节点处理环节;在失效修复阶段,新的通信任务不再被接受,已经接受和正在运行的任务将会终止,链路修复后由零负载开始接受新通信任务,因此通信链路的有效吞吐量和利用率将下降,平均响应时间将延长.

基于排队 Petri 网(QPN)的服务构件链路模型如图 6 所示.白色方框表示时间变迁(timed transition),黑色方框图代表瞬时变迁(immediate transition).圆圈表示普通库所(ordinary place),带有竖线的圆圈表示队列库所(queueing place).托肯(token)m 代表管理任务,包括在可用状态下执行的通信任务 l 和发生失效的通信任务 lf.标识作用的托肯 a 代表通信链路的状态(1=有效,0=无效),r 代表修复环节的完成情况(1=完成,0=进行中);控制作用的托肯 f_1 和 f_2 分别用于决定变迁 lt_7 和 lt_8 是否可以触发(0=禁止,1=允许).

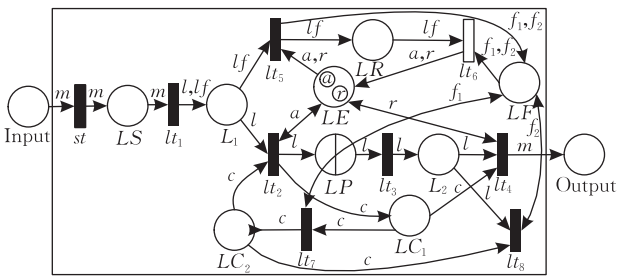

图 6　服务构件链路的 QPN 表示

图 6 中的变迁和库所描述如下.

(1)库所

库所 Input 和 Output 分别表示服务构件的管理任务输入和输出;

LS 表示进入的管理任务被链路接受;

L_1 用来接受按照一定的失效概率发生后形成的任务序列;

LE 用来限制链路资源 LP 在有效和失效修复两个阶段的使用是互斥和独占的，其内部托肯包括 a 和 r，取值在 0 和 1 之间切换；

LR 表示管理任务遇到链路失效后进入失效修复阶段；

LP 是多类队列库所，表示通信链路对信息传输的处理，包含一个 G/M/1/FCFS 队列；

L_2 用来接受 LP 完成的通信任务 l，链路有效时 l 直接转移到 Output 库所中，失效时记录任务 l 的数目后用来抵消链路修复后新接受的等量任务，以便模拟修复后的链路从零负载开始运行的效果；

LC_1 记录链路有效时进入的任务数目，接受新任务时数目增加，任务通信完成后数目减少；

LC_2 刻画了链路修复后用来抵消新进入链路通信任务的数目，在数值上等于失效时刻已经接受但被迫中断的任务数目 $num1$ 和修复期间 LP 完成的任务数目 $num2$ 的差($num1-num2$)；

LF 用来标识链路是否处于失效状态以便控制变迁 lt_7 和 lt_8 的触发.

(2) 变迁

瞬时变迁 st 表示管理任务进入到服务构件链路；

瞬时变迁 lt_1 表示管理任务按两种可能方式完成通信，一种是链路有效情况下直接进入链路队列，一种是失效修复后再进入链路队列，不同方式的选择由式(5)和(6)计算的链路失效概率 φ_L 决定；

瞬时变迁 lt_2 实现两种触发模式，完成的工作包括接受有效任务进入链路 LP，链路修复后用 LC_2 中记录的任务数目抵消新接受的等量任务；

瞬时变迁 lt_3 完成管理任务的数据通信操作；

瞬时变迁 lt_4 将完成的管理任务送到下一个处理环节；

瞬时变迁 lt_5 表示失效任务进入链路修复阶段，并设置控制用的托肯 f_1 和 f_2 为 1；

时间变迁 lt_6 表示链路经历一个失效修复过程，lt_6 的时间延迟服从 γ_L 的指数分布；

瞬时变迁 lt_7 将链路失效时刻已经接受但被迫中断的任务数目转移到 LC_2 中；

瞬时变迁 lt_8 用链路失效期间 LP 完成的任务数目抵消转移到 LC_2 中.

基于排队 Petri 网的服务构件节点模型在链路模型的基础上增加了对外部负载的描述，如图 7 所示. 托肯 m 代表管理任务，包括在可用状态下执行的节点任务 n 和发生失效的节点任务 nf，w 代表管理任务之外的工作负载. 托肯 a 代表处理节点的状态(1=有效，0=无效)，r 代表修复环节的完成情况(1=完成，0=进行中)，f 代表进入的节点是否有效(0=有效，1=失效)，ew 控制外部负载的产生(0=禁止，1=允许).

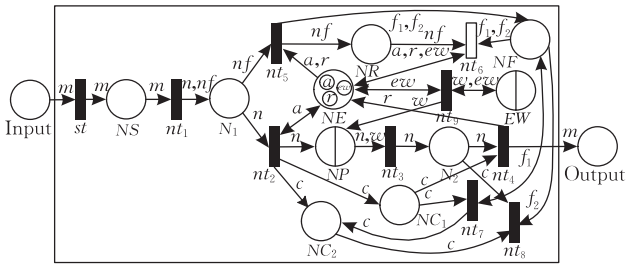

图 7 服务构件节点的 QPN 表示

与链路模型相比，主要的区别如下：

NP 是多类队列库所，表示服务节点对管理任务的执行，包含一个 G/M/m/PS 队列；

EW 是队列库所，产生管理任务之外的工作负载，包含一个 G/M/∞/IS 队列；

瞬时变迁 nt_1 表示管理任务在数据交换成功后以两种可能方式进入到节点处理队列中执行：一种是直接处理，一种是节点失效修复后再由节点处理，不同方式的选择由式(4)计算的服务构件节点的失效概率 φ_{ki} 决定；

时间变迁 nt_6 表示节点经历一个失效修复过程，nt_6 的时间延迟服从($\gamma_p+\gamma_n$)的指数分布；

瞬时变迁 nt_9 在托肯 ew 的控制下产生额外工作负载进入到服务构件节点中.

4.4.2 建模管理服务执行力

服务平台管理服务的组合是应用驱动的，服务构件组成和执行序列由应用需求、管理目标和自管理策略决定. 为此，构造一个应用场景来说明管理服务的执行过程，如图 8 所示. 具体为：(1) 用户管理服务构件(Usc)处理用户的访问控制；(2) 调用应用管理服务构件(Asc)完成用户应用请求的预处理；(3) 调用资源管理服务构件(Rsc)完成资源分配和任务调度，由于需要进行协商，所以可能需要重复多次运行 Rsc 构件，假设为 k 次；(4) 调用两个执行管理服务构件(Esc)进行并发任务的处理；(5) 将处理结果传送给 Usc 构件，然后按照用户需求显示给用户. 该执行过程可以形式化表示为“Usc；Asc；(ηRsc)；($Esc_1 \parallel Esc_2$)；Usc”.

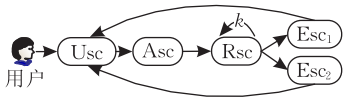

图 8 管理服务示例的服务构件运行图

该应用场景对应的排队 Petri 网模型如图 9 所示. 队列库所 Q 表示用户请求队列，库所 U 表示请求任务等待被链路接受，库所 P_i 表示管理任务 t 完成上一个阶段的工作后等待进入下一阶段处理. 加粗的带线圆圈 L_i 和 N_i 分别表示链路和节点的排队 Petri 网. 所有被调用的服务构件可能分布在多个硬件资源上，服务构件的处理能力依赖于处理器个数.

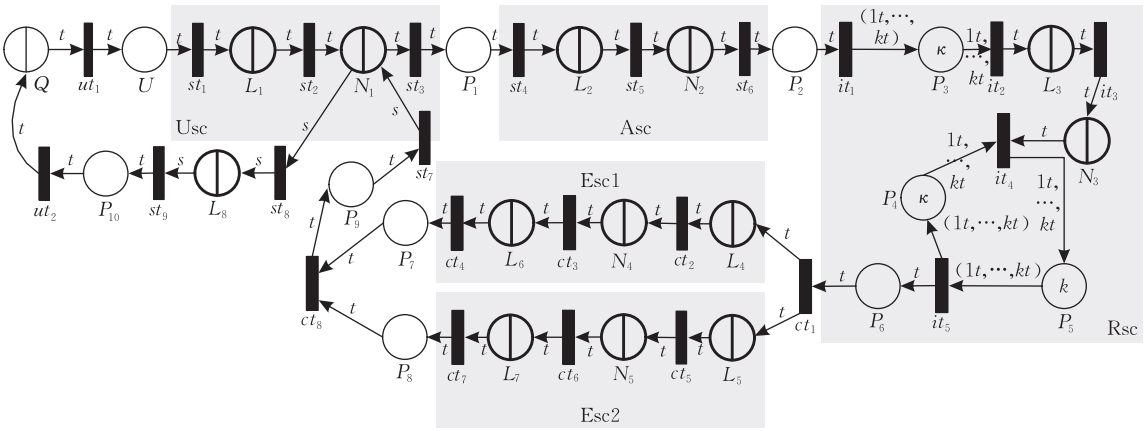

图 9　管理服务示例的 QPN 表示

通过 Petri 网的变迁触发机制的设计可以描述服务构件之间多种交互关系，瞬时变迁 st_i，it_i 和 ct_i 分别为顺序、重复和并发方式下的过程变迁. 图 9 中变迁的设计如表 3 所示，符号“$A\{x\}\rightarrow B\{y\}$”表示模型中变迁的触发模式(firing mode)，托肯“x”在一次变迁中由库所 A 迁出后以“y”的形式存放在库所 B 中. 符号“$A\{x\}\rightarrow\{\}$”表示“x”在变迁中由库所 A 迁出后不留在任何库所中，而是被销毁(destroy).

表 3　管理服务变迁触发模式表

变迁	触发模式
ut_1, ut_2	1：$In\{t\}\rightarrow Out\{t\}$
$st_1, st_2, st_3, st_4, st_5, st_6$，	1：$In\{t\}\rightarrow Out\{t\}$
st_7	1：$P_9\{t\}\rightarrow N_1\{s\}$
st_8	1：$N_1\{s\}\rightarrow L_8\{s\}$
st_9	1：$L_8\{s\}\rightarrow P_{10}\{t\}$
it_1	1：$P_2\{t\}\rightarrow P_3\{1t\}+\cdots+P_3\{kt\}$
it_2	1：$In\{1t\}\rightarrow Out\{t\}$ 2：$In\{2t\}\rightarrow Out\{t\}$ … k：$In\{kt\}\rightarrow Out\{t\}$
lt_3	1：$L_3\{t\}\rightarrow N_3\{t\}$
it_4	1：$P_4\{1t\}+N_3\{t\}\rightarrow P_5\{t\}$ 2：$P_4\{2t\}+N_3\{t\}\rightarrow P_5\{t\}$ … k：$P_4\{kt\}+N_3\{t\}\rightarrow P_5\{t\}$
it_5	1：$P_5\{1t\}+\cdots+P_5\{kt\}\rightarrow$ $P_6\{t\}+P_4\{1t\}+\cdots+P_4\{kt\}$
ct_1	1：$P_6\{t\}\rightarrow L_4\{t\}+L_5\{t\}$
$ct_2, ct_3, ct_4, ct_5, ct_6, ct_7$	1：$In\{t\}\rightarrow Out\{t\}$
ct_8	1：$P_7\{t\}+P_8\{t\}\rightarrow P_9\{t\}$

5　实验与分析

5.1　参数配置

本文采用 QPME 1.01 软件包[43-44]计算稳态时模型中每个库所的吞吐量、平均逗留时间以及平均标识的数量，预测服务构件和管理服务的平均响应时间和利用率. 服务请求的发生时间一般可设为随机变量，运行服务构件的节点队列根据服务管理请求的到达先后确定对象的处理序列，服务策略是预先确定的，信源分布也是事先确定的. 由于 QPME 现有版本还不支持时间变迁的直接使用，本文的仿真实验采用瞬时变迁和队列库所组合等价替换时间变迁 lt_6 和 nt_6，完成的服务构件 QPN 模型如图 10 所示. 新增加的队列库所 Q 模拟稳定的管理任务产生源，LR' 表示通信链路失效后等待失效修复处理，NR' 表示节点软硬件失效后管理任务等待失效修复处理，库所 $Link$、$Node$ 和 SC 分别用来跟踪和描述链路、节点软硬件和服务构件的总体执行状况.

由于服务管理请求的到来服从泊松分布，服务请求的处理时间、失效修复时间服从指数分布，所以库所属性如表 4 所示.

图 10 中变迁的设计如表 5 所示，对于 lt_1 和 nt_1 这样的变迁属于自由选择冲突模型，选择哪一个变迁模式实施并不依赖于库所中的标识，而取决于变迁模式的触发权值(firing weight). 令变迁 lt_1 的变迁模式的触发权值分别为 $w(1)$ 和 $w(2)$，且 $w(2)=1$，那么 $\varphi_L=\dfrac{w(2)}{w(1)+w(2)}=\dfrac{1}{w(1)+1}$，所以 $w(1)$ 与链路失效概率 φ_L 成近似倒数关系. 同理，可以求得变迁 st_1 的变迁模式的触发权值. 变迁 lt_3 和 nt_3 的设计将有效和失效管理任务区别对待，有效任务进入到后续环节(mode1)，无效任务转化为一定的负载(mode2).

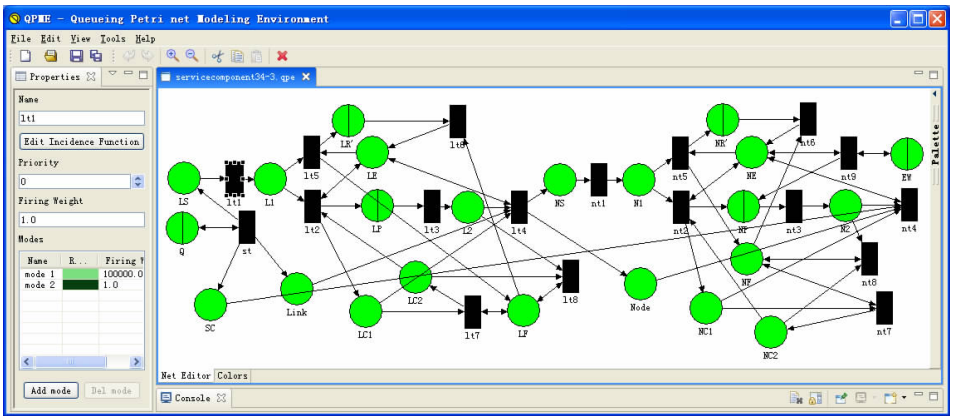

图 10 服务构件的 QPN 模型

表 4 库所属性表

库所	托肯	初始值	队列类型	离队规则	参数 p_1
Q	$\{m\}$	$\{1\}$	G/M/∞/IS	Normal	0.0001
LS,NS	$\{m\}$	$\{0\}$	na	FIFO	
LE,NE	$\{a,r\}$	$\{1,1\}$	na	Normal	
LF,NF	$\{f_1,f_2\}$	$\{0,0\}$	na	Normal	
$LC_1,LC_2,$ NC_1,NC_2	$\{c\}$	$\{0\}$	na	Normal	
L_1	$\{l,lf\}$	$\{0,0\}$	na	FIFO	
L_2	$\{l\}$	$\{0\}$	na	FIFO	
LP	$\{l\}$	$\{0\}$	G/M/1/FCFS	Normal	0.005
LR'	$\{lf\}$	$\{0\}$	G/M/∞/IS	Normal	0.0001
N_1	$\{n,nf\}$	$\{0,0\}$	na	FIFO	
N_2	$\{n\}$	$\{0\}$	na	FIFO	
NP	$\{n,w\}$	$\{0,0\}$	G/M/2/PS	Normal	$\{0.0001,0.0001\}$
NR'	$\{nf\}$	$\{0\}$	G/M/∞/IS	Normal	0.00001
EW	$\{w\}$	$\{1\}$	G/M/∞/IS	Normal	0.000005
$Link,Node,SC$	$\{m\}$	$\{0\}$	na	Normal	

表 5 服务构件变迁触发模式表

变迁	触发模式
st	1: $Q\{m\}\rightarrow Link\{m\}+SC\{m\}+LS\{m\}+Q\{m\}$
lt_1	1: $LS\{m\}\rightarrow L_1\{l\}$ 2: $LS\{m\}\rightarrow L_1\{lf\}$
lt_2	1: $L_1\{l\}+LE\{a\}\rightarrow LP\{l\}+LE\{a\}+LC_1\{c\}$ 2: $L_1\{l\}+LE\{a\}+LC_2\{c\}\rightarrow LE\{a\}+LC_1\{c\}$
lt_3	1: $LP\{l\}\rightarrow L_2\{l\}$
lt_4	1: $L_2\{l\}+LE\{r\}+LC_1\{c\}+Link\{m\}\rightarrow$ $NS\{m\}+Node\{m\}+LE\{r\}$
lt_5	1: $L_1\{lf\}+LE\{a\}+LE\{r\}\rightarrow LR'\{lf\}+LF\{f_1\}+LF\{f_2\}$
lt_6	1: $LR'\{lf\}+LF\{f_1\}+LF\{f_2\}\rightarrow LE\{a\}+LE\{r\}$
lt_7	1: $LC_1\{c\}+LF\{f_1\}\rightarrow LC_2\{c\}+LF\{f_1\}$
lt_8	1: $LC_2\{c\}+LF\{f_2\}+L_2\{l\}\rightarrow LF\{f_2\}$
nt_1	1: $NS\{m\}\rightarrow N_1\{n\}$ 2: $NS\{m\}\rightarrow N_1\{nf\}$
nt_2	1: $N_1\{n\}+NE\{a\}\rightarrow NP\{n\}+NE\{a\}+NC_1\{c\}$ 2: $N_1\{n\}+NE\{a\}+NC_2\{c\}\rightarrow NE\{a\}+NC_1\{c\}$
nt_3	1: $NP\{n\}\rightarrow Q\{m\}+N_2\{n\}$ 2: $EW\{w\}\rightarrow\{\}$
nt_4	1: $N_2\{n\}+SC\{m\}+Node\{m\}+NC_1\{c\}+NE\{r\}\rightarrow NE\{r\}$
nt_5	1: $N_1\{nf\}+NE\{a\}+NE\{r\}+NE\{ew\}\rightarrow$ $NF\{f_1\}+NF\{f_2\}+NR'\{nf\}$
nt_6	1: $NF\{f_1\}+NF\{f_2\}+NR'\{nf\}\rightarrow$ $NE\{a\}+NE\{r\}+NE\{ew\}$
nt_7	1: $NC_1\{c\}+NF\{f_1\}\rightarrow NC_2\{c\}+NF\{f_1\}$
nt_7	1: $NC_2\{c\}+NF\{f_2\}+N_2\{n\}\rightarrow NF\{f_2\}$
nt_8	1: $EW\{w\}+NE\{ew\}\rightarrow NE\{ew\}+NP\{w\}+EW\{w\}$

5.2 数据分析

5.2.1 失效率和修复时间的影响

本文分两个场景讨论.

场景 1. 初始时,变迁 lt_1 和 st_1 的 $w(1)$ 都取值 100000,$w(2)$ 都取值 1,表 4 中其它变迁及变迁模式的触发权值也设置为 1,库所的队列类型和参数 p_1 见表 3,然后假设其它参数不变,只有链路失效概率 φ_L 即变迁 lt_1 的 $w(1)$ 发生变化.

场景 2. 初始时,变迁 lt_1:$w(1)=100000$,变迁 st_1:$w(1)=1000$,$w(2)$ 取值 1,NR':$p_1=0.00001$,然后假设其它参数不变,只有节点失效修复时间(即库所 NR' 的参数 p_1)发生变化. 仿真结果见表 6,置信区间为 95%,性能属性包括稳态时的到达率(A_i)、负载量(X_i)、利用率(U_i)和平均响应时间(τ_i).

场景 1 中,在稳定任务源(A_Q)的驱动下,随着 $w(1)$ 由 100000 减小到 1000,链路失效率依次增加,导致链路失效修复环节(LR')的到达率($A_{LR'}$)、负载量($X_{LR'}$)和利用率($U_{LR'}$)逐渐增加,而通信环节(LP)的任务到达率(A_{LP})、负载量(X_{LP})和利用率(U_{LP})下降,有效任务的吞吐量减少,越来越多的任务(X_{LC_2})由于失效在运行期间被中断. 日益频繁的失效扩大了修复环节的时间延迟对链路的影响,使得管理任务在链路($Link$)中的平均逗留时间(τ_{Link})变长,单位时间内逗留在链路中的任务数量(X_{Link})快速上升,进一步影响了服务构件的整体性能,负载量(X_{SC})和平均响应时间(τ_{SC})都被迫增加.

场景 2 中,链路和节点的失效率保持不变,保证了节点的任务到达率(A_{NS})的稳定. 随着修复环节(NR')的参数 p_1 由 0.00001 增大为 0.001,节点失效修复的时间($\tau_{NR'}$)逐渐变少,一方面使得节点修复环节(NR')的处理能力增强,负载量($X_{NR'}$)降低,另一方面又导致节点的有效运行时间相对增加,有

效任务到达率($A_{NP}(n)$)提高，节点处理能力(X_{Node})增强，有效任务负载量($X_{NP}(n)$)和额外负载($X_{NP}(w)$)下降. NC_2刻画了节点修复后抵消新进入的任务的数目，在数值上等于失效时刻逗留在 NP 中的任务数目 $num1$ 和修复期间 NP 完成的任务数目 $num2$ 的差($num1-num2$). 修复时间的缩短使得 $num2$ 的值减少，因此 NC_2 的负载量(X_{NC_2})和利用率(U_{NC_2})都提高了. 对于节点来讲，节点处理能力(X_{Node})的提高减少了任务完成时间(τ_{Node})，进而影响到服务构件的整体性能(X_{SC}，τ_{SC}).

表 6　仿真结果比较

场景 1 (LR'：$p_1=0.0001$，LP：$p_1=0.005$)				场景 2 (st_1：$w(1)=1000$，NP：$p_1=0.0001$)			
lt_1：$w(1)$	100000	10000	1000	NR'：p_1	0.00001	0.0001	0.001
A_Q	9.9981E-5	9.9878E-5	9.9875E-5	A_{NS}	9.9933E-5	1.0011E-4	1.0014E-4
$A_{LR'}$	5.0000E-10	8.5001E-9	1.0030E-7	$A_{NR'}$	9.6101E-8	1.02201E-7	9.4801E-8
$X_{LR'}$	3.8968E-6	9.5544E-5	0.001005	$X_{NR'}$	0.009682	0.001029	9.8402E-5
$U_{LR'}$	3.8968E-6	9.5544E-5	0.001005	$\tau_{NR'}$	100754.8891	10077.7377	1037.9824
A_{LP}	9.9980E-5	9.9869E-5	9.9775E-5	$A_{NP}(n)$	9.981461E-5	9.992573E-5	9.992582E-5
X_{LP}	0.02041	0.02039	0.02038	$X_{NP}(n)$	1.4707	1.3814	1.3697
U_{LP}	0.02001	0.01999	0.01996	$X_{NP}(w)$	0.07267	0.06925	0.06917
X_{LC_1}	0.02041	0.02039	0.02038	X_{NC_1}	1.4685	1.3793	1.3673
U_{LC_1}	0.02001	0.01999	0.01996	U_{NC_1}	0.668699	0.671392	0.670217
X_{LC_2}	0.0	2.3308E-8	2.6877E-7	X_{NC_2}	0.002898	0.003662	0.004686
U_{LC_2}	0.0	2.3308E-8	2.5887E-7	U_{NC_2}	0.0011588	0.001805	0.0022517
X_{Link}	1.6051	39.7711	488.5048	X_{Node}	1253.1374	1152.1419	1151.2348
τ_{Link}	16047.5035	398055.2318	4891007.2996	τ_{Node}	1.25396E7	1.150867E7	1.1497E7
X_{SC}	15.6016	50.5698	497.5156	X_{SC}	1258.5566	1160.9329	1155.2123
τ_{SC}	156044.7427	506098.9915	4981225.7033	τ_{SC}	1.2594E7	1.15963E7	1.1537E7

5.2.2　自主性的作用

在图 10 建模的基础上，赋予服务构件一定的自适应能力，即能够及时发现任务失效并立即重新调度管理任务的执行，那么得到扩展的服务构件 QPN 模型如图 11 所示，新增的库所 LC_3 和 NC_3 分别用来记录链路和节点失效时刻已经接受但被迫中断的任务数目.

变迁模式的改变如表 7 所示.

变迁 lt_1 和 st_1 的 $w(1)$ 都取值 1000，$w(2)$ 都取值 1，表 5 中其它变迁及变迁模式的触发权值也设置为 1，库所的队列类型和参数 p_1 见表 4，两个模型的度量结果对比见表8. 在相同的初始配置条件下，引入自适应失效重调度机制后，链路、节点和服务构件的到达率下降，负载量(X_i)、利用率(U_i)和平均响应时间(τ_i)优化明显.

表 7　变迁触发模式表

变迁	触发模式
lt_5	1：$L_1\{lf\}+LE\{a\}+LE\{r\}\rightarrow LR'\{lf\}+LF\{f_1\}+LF\{f_2\}+LS\{m\}$
lt_7	1：$LC_1\{c\}+LF\{f_1\}\rightarrow LC_2\{c\}+LF\{f_1\}+LC_3\{c\}$
lt_9	1：$LC_3\{c\}\rightarrow LS\{m\}$
nt_5	2：$N_1\{nf\}+NE\{a\}+NE\{r\}+NE\{ew\}\rightarrow NF\{f_1\}+NF\{f_2\}+NR'\{nf\}+NS\{m\}$
nt_7	1：$NC_1\{c\}+NF\{f_1\}\rightarrow NC_2\{c\}+NF\{f_1\}+NC_3\{c\}$
nt_{10}	1：$NC_3\{c\}\rightarrow NS\{m\}$

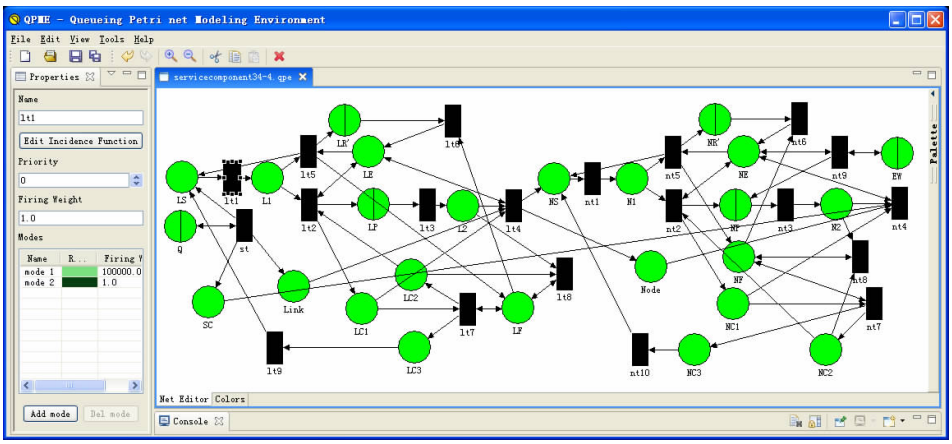

图 11　扩展的服务构件 QPN 模型

表 8 度量结果对比

度量	模型 1	模型 2	度量	模型 1	模型 2	度量	模型 1	模型 2
A_{Link}	1.0014E-4	9.9998E-5	A_{Node}	1.00036E-4	9.9998E-5	A_{SC}	1.001434E-4	9.999808E-5
X_{Link}	527.2974	0.02235	X_{Node}	1347.0343	1.6544	X_{SC}	1874.3317	1.67678
U_{Link}	0.999945	0.020951	U_{Node}	0.999865	0.680174	U_{SC}	0.999987	0.687062
τ_{Link}	5265147.3879	223.4963	τ_{Node}	1.3466E7	16544.5983	τ_{SC}	1.87165E7	16768.09496

由结果可知，要想获得满意的性能，一方面必须降低服务构件在链路和节点软硬件方面的失效率，减少发生故障的可能；另一方面，必须缩短修复时间，提高服务构件的失效修复能力.除了服务构件自身的优化，加强对失效发生的实时监控和自主处理能力显得更加重要.

5.2.3 容错的作用

提高可靠性的主要方法是冗余，包括设备冗余、信息冗余、时间冗余等.设备冗余包括硬件冗余和软件冗余.本文的执行力模型能够对服务构件冗余情况下的性能与可用性进行分析.为此，在图 9 管理服务示例的基础上对应用服务构件 *Asc* 进行冗余处理，对应的执行力模型如图 12 所示.

增加的变迁触发模式如表 9 所示.

表 9 新增的变迁触发模式表

变迁	触发模式
ft_1	1：$P_1\{t\}\rightarrow L_{21}\{t\}+L_{22}\{t\}+P_{21}\{n\}$
ft_2, ft_3, ft_4, ft_5	1：$In\{t\}\rightarrow Out\{t\}$
ft_6	1：$P_{21}\{n\}+P_{22}\{t\}\rightarrow P_2\{t\}+P_{24}\{b\}$
ft_7	1：$P_{21}\{n\}+P_{23}\{t\}\rightarrow P_2\{t\}+P_{25}\{b\}$
ft_8	1：$P_{24}\{b\}+P_{23}\{t\}\rightarrow\{\}$
ft_9	1：$P_{25}\{b\}+P_{22}\{t\}\rightarrow\{\}$

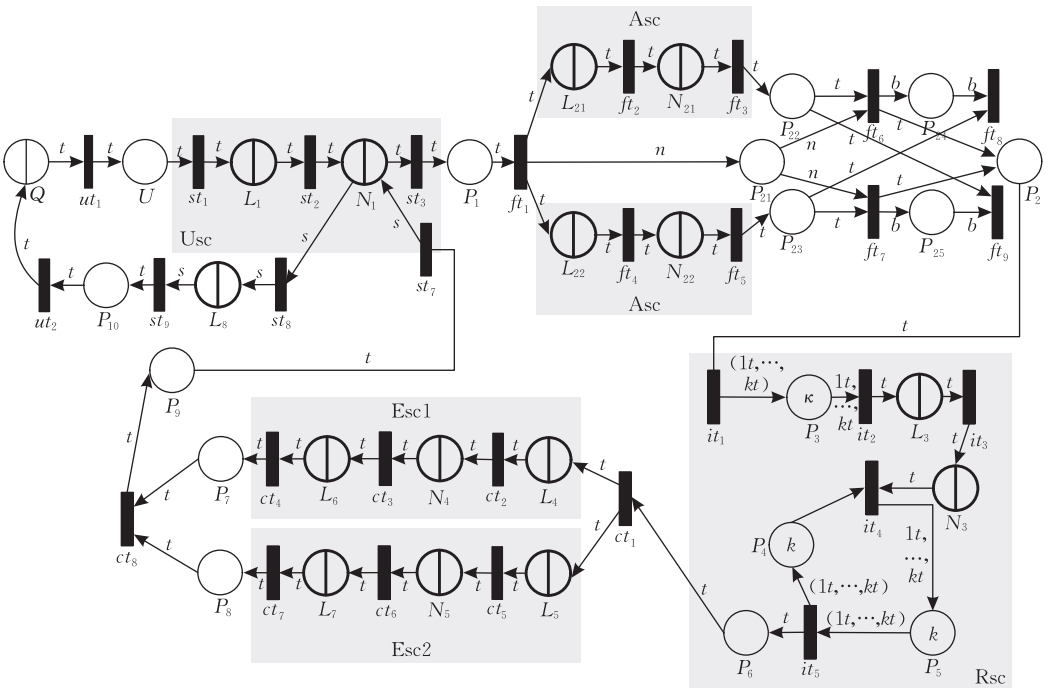

图 12 冗余处理后管理服务示例的 QPN 表示

实验结果表明，经过冗余处理后管理服务的可靠性和性能都有所提高，其变化趋势和理论分析是一致的.此外，提出的执行力建模方法也能很好地反映服务构件组成和交互方式对管理服务运行效果的影响.

6 结论和下一步工作

随着信息技术的迅猛发展，大规模、开放、异构、动态的信息服务系统不断涌现，如网格计算、云计算和普适计算等.这些系统越来越倾向于构建面向服务的计算和资源管理平台，实现功能软件实体的敏捷部署，并能根据环境变化和用户需求对服务平台进行自我管理.但是从已有的研究成果[3]看，自我管理还未形成系统的、成熟的理论体系，所涉及到的技术问题也是多方面多层次的，还需要更深入的研究.

本文提出的自管理服务平台参考模型以构件和服务作为基本组成方式，将构件的紧耦合和服务的松耦合结合起来，以构件的形式实现功能，以服务的形式实现交互，形成可伸缩的自动化中间件管理模

式.通过设计基于分层反馈的自主控制框架提高了服务平台自管理的可扩展性和灵活性,能够保证各个组成部件能够有机地结合起来形成高效、有序的虚拟计算环境.针对面向 Internet 的服务平台中资源故障和构件失效多发的情况,以综合考虑了可用性和性能的执行力模型作为自主控制的决策基础,有助于把握服务平台的内部结构和外部环境的不确定特性,并保持服务平台运行的高性能.服务平台执行力模型既涉及到软硬件自身的运行状态,也受制于多个服务构件的交互关系和组织结构的选择.执行力建模过程中运用了马尔可夫过程、随机 Petri 网和排队论,有利于对服务平台的非功能特性进行定性和定量分析.仿真结果表明基于排队 Petri 网的执行力模型能够反映失效率和修复能力对性能和可用性的影响,并验证了自主控制方法对提高管理服务执行力的积极作用.

为了更好地说明所提出方法的可行性,我们在已经完成的国家"十一五"科技支撑计划项目子课题"数字教育资源集成与共享"的系统成果基础上研制了相关原型系统.数字教育资源集成与共享系统平台采用基于构件和面向服务的思想,选择 IBM Rational 进行服务构件和业务服务的设计、开发和封装,并采用 WebSphere 进行服务平台的部署.服务架构采用对内和对外两级分布式结构,其中内部服务包括副本管理服务、目录管理服务、存储调度服务与传输控制服务,对外服务包括资源标准化服务、资源注册服务、统计分析服务、资源访问服务、资源管理服务以及信息检索服务.原型系统的设计思路是利用所提出的方法对该系统平台的执行力进行建模,并通过在线仿真工具 QPME 对所部属服务的执行力进行预测与评估,然后结合服务构件的配置、诊断、修复和重配置技术,实现系统平台的自主控制.已经完成的工作表明所提出的方法能够有效提高系统平台的自管理能力,但限于篇幅,具体细节将另文阐述.

在下一步的工作中,我们将针对服务平台的功能性和非功能性需求,设计合适的自主控制策略,探讨服务构件的性能和可用性监控技术,并在此基础上对原型系统进行扩展,进一步验证其有效性并进行改进.

致　谢　衷心感谢 Samuel Kounev 博士提供的排队 Petri 网仿真软件 QPME 1.01 对本文研究工作的帮助!

参　考　文　献

[1] Papazoglou M P, Georgakopoulos D. Service oriented computing. Communications of the ACM, 2003, 46(10): 24-28

[2] Foster I, Kesselman C, Nick J et al. The physiology of the grid: An open grid services architecture for distributed systems integration. Global Grid Forum, 2002

[3] Liao Bei-Shui, Li Shi-Jian, Yao Yuan, Gao Ji. Research on conceptual model and realization methods of autonomic computing, Journal of Software, 2008, 19(4): 779-802(in Chinese)
(廖备水,李石坚,姚远,高济.自主计算概念模型与实现方法研究.软件学报,2008,19(4):779-802)

[4] van Moorsel A P A. Grid, management and self-management. The Computer Journal, 2005, 48(3): 325-332

[5] Gu Jun, Luo Jun-Zhou. Reference model for self-managing and light-weighting grid platform. Journal of Southeast University (Natural Science Edition), 2008, 38(Sup. I): 185-191(in Chinese)
(顾军,罗军舟.一种自管理的轻量化网格平台参考模型.东南大学学报(自然科学版),2008,38(增刊 I):185-191)

[6] Meyer J F. On evaluating the performability of degradable computing systems. IEEE Transactions on Computers, 1980, 29(8): 720-731

[7] Horn P. Autonomic computing: IBM's perspective on the state of information technology. IBM Corporation, 2001. http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf

[8] Kephart J O, Chess D M. The vision of autonomic computing. IEEE Computer, 2003, 36(1): 41-50

[9] Kephart J O, Walsh W E. An artificial intelligence perspective on autonomic computing policies//Verma D, Devarakonda M, Lupu E, Kohli M. Proceedings of the 5th IEEE International Workshop on Policies for Distributed Systems and Networks. New York: IEEE Computer Society, 2004: 3-12

[10] Kephart J O, Das R. Achieving self-management via utility functions. IEEE Internet Computing, 2007, 11(1): 40-47

[11] Hinchey M G, Sterritt R. Self-managing software. Computer, 2006, 39(2): 107-109

[12] Aldinucci M, Danelutto M, Vanneschi M. Autonomic QoS in ASSIST grid-aware components//Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP'06). Montbéliard, France, 2006: 221-230

[13] Aldinucci M, Campa S, Danelutto M et al. Behavioral skeletons in GCM: Autonomic management of grid components//Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP'08). Toulouse, France, 2008: 54-63

[14] Parashar M, Li Z, Liu H et al. Enabling autonomic grid applications: Requirements, models and infrastructures//Babaoglu O et al. Proceedings of the International Workshop on Self-* Properties in Complex Information Systems. Berlin: Springer Verlag, 2005: 273-290

[15] Parashar M, Liu H, Li Z et al. AutoMate: Enabling autonomic applications on the grid. Cluster Comput, 2006, 9: 161-174

[16] Augusto M, Sallem S, de Sousa S A, da Silva e Silva F J. AutoGrid: Towards an autonomic grid middleware//Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2007). 2007: 223-228

[17] Van Roy P, Ghodsi A, Haridi S et al. Self management of large-scale distributed systems by combining Peer-to-Peer networks and components//Proceedings of the 2nd CoreGRID Workshop on Grid and Peer to Peer Systems Architecture. Paris, 2006

[18] Zhan Jian-Feng, Wang Lei, Zou Ming et al. Grid unit: A self-managing building block for grid system//Proceedings of the 8th International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, 2007: 303-310

[19] Kaufman J, Lehman T, Deen G, Thomas J. OptimalGrid — autonomic computing on the grid. IBM developerWorks, 2003

[20] van Moorsel A P A. Grid, management and self-management. The Computer Journal, 2005, 48(3): 325-332

[21] Haverkort B R, Marie R, Rubino G, Trivedi K S. Performability Modelling: Techniques and Tools. John Wiley & Sons, Inc. 2001

[22] Wang Yuan-Zhuo, Lin Chuang, Yang Yang, Shan Zhi-Guang. Research on manageability of grid service model method and managemen strategies. Chinese Journal of Computers, 2008, 31(10): 1716-1726(in Chinese)
(王元卓，林闯，杨扬，单志广. 网格服务可管理性模型及策略研究. 计算机学报，2008，31(10)：1716-1726)

[23] Ramani S, Trivedi K S, Dasarathy B. Performance analysis of the CORBA Event Service using stochastic reward nets//Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems. Nurnberg, Germany, 2000: 238-247

[24] Ramani S, Trivedi K S, Dasarathy B. Performance analysis of the CORBA notification service//Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems. New Orleans, USA, 2001: 227-236

[25] Ramani S, Goseva-Popstojanova K, Trivedi K S. A framework for performability modeling of messaging services in distributed systems//Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems. Greenbelt, MD, 2002: 229-238

[26] Trivedi K S, Ramani S, Fricks R. Recent advances in modeling response-time distributions in real-time systems. Proceedings of the IEEE,2003, 91(7): 1023-1037

[27] Liu Xian-Ming, Li Shi-Xian. A performability model for grid monitoring architecture. Computer Science, 2005, 32(1): 63-67, 99(in Chinese)
(刘显明，李师贤. 网格监控体系结构的一种可执行性模型. 计算机科学，2005，32(1)：63-67，99)

[28] Das O, Woodside C M. Dependable-LQNS: A performability modeling tool for layered systems//Proceedings of the 13th International Conference on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS 2003). Urbana, Illinois, USA, 2003: 672

[29] Das O, Woodside M. Computing the performability of layered distributed systems with a management architecture//Proceedings of the 4th International Workshop on Software and Performance (WOSP 04). Redwood Shores, California, 2004: 174-185

[30] Qu Yang, Lin Chuang, Wang Yuan-Zhuo, Shan Zhi-Guang. Performability evaluation of resource scheduling algorithms for computational grids//Proceedings of the 5th International Conference on Grid and Cooperative Computing (GCC06). IEEE Computer Society, 2006: 458-465

[31] Kogekar A, Kaul D, Gokhale A et al. Model-driven generative techniques for scalable performabality analysis of distributed systems//Proceeding of the 20th IEEE International Parallel & Distributed Processing Symposium(IPDPS 2006). Rhodes Island, Greece, 2006: 336-343

[32] Bernardi S, Bobbio A, Donatelli S. Petri nets and dependability. Lectures on Concurrency and Petri Nets, 2003: 125-179

[33] Lin Chuang, Wang Yuan-Zhuo, Yang Yang, Qu Yang. Research on network dependability analysis methods based on stochastic Petri net. Acta Electronica Sinica, 2006, 34(2): 322-332(in Chinese)
(林闯，王元卓，杨扬，曲扬. 基于随机 Petri 网的网络可信赖性分析方法研究. 电子学报，2006，34(2)：322-332)

[34] Bause F. Queueing Petri nets—A formalism for the combined qualitative and quantitative analysis of systems//Proceedings of the 5th International Workshop on Petri Nets and Performance Models. Toulouse, France, 1993: 14-23

[35] Kounev S, Buchmann A. Performance modelling of distributed E-business applications using queueing Petri nets//Proceedings of the 2003 IEEE International Symposium Performance Analysis of Systems and Software. Austin, Texas, 2003: 143-155

[36] Lu Jian, Ma Xiao-Xing, Tao Xian-Ping et al. Research and development of interware. Science in China (Series E: Information Sciences), 2006, 36(10): 1037-1080(in Chinese)
(吕建，马晓星，陶先平等. 网构软件的研究与进展. 中国科学 E 辑(信息科学)，2006，36(10)：1037-1080)

[37] Dai Ru-Wei, Cao Long-Bing. Internet—A open complex giant system. Science in China (Series E), 2003, 33(4): 289-296(in Chinese)
(戴汝为，操龙兵. Internet——一个开放的复杂巨系统. 中国科学(E 辑)，2003，33(4)：289-296)

[38] Hu Hai-Yang. Reliability analysis for component-based software system in open distributed environments. IJCSNS International Journal of Computer Science and Network Security, 2007, 7(5): 193-202

[39] Jarvis S, Thomas N, van Moorsel A. Open issues in grid performability. International Journal of Simulation: Systems, Science and Technology, 2004, 5(5): 3-12

[40] Mao Xiao-Guang, Deng Yong-Jin. A general model for component-based software reliability. Journal of Software, 2004,

15(1)：27-32(in Chinese)
(毛晓光，邓勇进. 基于构件软件的可靠性通用模型. 软件学报，2004，15(1)：27-32)
[41] Yang B，Xie M. A study of operational and testing reliability in software reliability analysis. Reliability Engineer and System Safety，2000，70：323-329
[42] Guo Su-Chang，Yang Bo，Huang Hong-Zhong. Modeling and analysis for grid service reliability considering node recovery. Journal of Xi′an Jiaotong University，2008，42(6)：693-697，790(in Chinese)
(郭夙昌，杨波，黄洪钟. 考虑节点失效恢复能力的网格服务可靠性建模与分析. 西安交通大学学报，2008，42(6)：693-697，790)
[43] Kounev S，Dutz C，Buchmann A. QPME—Queueing Petri net modeling environment//Proceedings of the 3rd International Conference on Quantitative Evaluation of SysTems (QEST 2006). Riverside，CA，2006：115-116
[44] Kounev S，Buchmann A. SimQPN—A tool and methodology for analyzing queueing Petri net models by means of simulation. Performance Evaluation，2006，63(4-5)：364-394

GU Jun，born in 1977，Ph. D. candidate，lecturer. His main research interests include network and service computing.

LUO Jun-Zhou，born in 1960，Ph. D.，professor，Ph. D. supervisor. His research interests include next generation network architecture，protocol engineering，network security，grid and cloud computing，and wireless local area network.

CAO Jin-Xin，born in 1967，Ph. D.，professor，Ph. D. supervisor. His research interests include network security，mobile network and service computing，information transmission behavior in complex network.

LI Wei，born in 1978，Ph. D.，associate professor. His research interests include next generation network，network management and service computing.

Background

This work is supported by the National Basic Research Program of China (973 Program) under Grant No. 2010CB328104，the National Nature Science Foundation of China under Grant Nos. 60903161，60903162，61003257，61070161，61070158，61003311，Research Fund for the Doctoral Program of Higher Education of China under Grant Nos. 200802860031，20110092130002，Jiangsu Provincial Natural Science Foundation of China (Key program) under Grants No. BK2008030，Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No. BM2003201，and Key Laboratory of Computer Network and Information Integration of the Ministry of Education under Grants No. 93K-9.

Traditionally，network and system management are manually controlled processes. It usually takes one or more human operators to manage all aspects of a dynamically evolving computing system. The operator is tightly integrated in this management process，and his or her tasks range from defining high-level policies to executing low-level system commands for immediate problem resolution. Although this form of human-in-the-loop management was appropriate in the past，it has become increasingly unsuitable for modern networked computing systems.

As the size，complexity and adaptability required by applications increasing，so does the need for software systems which are scalable，support dynamic composition and rigorous analysis，and are flexible and robust in the presence of change. Self-management is the really vision that the systems are capable of self-configuration，self-adaptation and self-healing，self-monitoring and self-tuning，and so on，often under the flag of self-* or autonomic systems. However，an issue in self-managed systems is that different abstractions and programming models are used on different architectural layers，leading to systems that are harder to build and understand. To alleviate this，we introduce a self-management service platform architectural reference model which is characterized by component-based，service-oriented and interaction-centric. Organized in a hierarchical autonomic control structure based on feedback mechanism this allows for self-management in distributed systems. The proposed approach takes the service components and their interaction relations as controlled objects，and makes decision according to the analytical performability model of service platform. A combination of availability and performance models of management service is suggested via Markov Procedure，Stochastic Petri Net and Queueing Network theory. The simulation results show that the proposed performability model based on Queueing Petri Net can quantitatively analyze the influence of failure probability and recovery time on availability and performance of service platform，and autonomic control approach is helpful for improving the effectiveness of service platform.