基于方面模板的分布式组件系统性能预测方法

黄 翔 张文博 张 波 魏 峻
(中国科学院软件研究所软件工程技术研发中心 北京 100080)

摘 要 目前分布式组件系统通常需要中间件提供的横切关注点(crosscutting)实现非功能性特征,而这种设计方法在简化分布式组件系统开发过程的同时难以在设计时进行有效的性能预测。研究了一种基于方面模板的分布式组件系统性能预测方法,将横切关注点提炼为可复用的方面模板,并通过面向方面的建模技术,自动构建包括中间件横切关注点和中间件性能因素的完整组件性能模型。该模型的预测结果可辅助设计人员尽早地发现组件设计缺陷或帮助筛选备选方案。

关键词 性能预测,中间件,面向方面,模板

Performance Predication of Distributed Component Systems Based on Aspect Templates

HUANG Xiang ZHANG Wen-bo ZHANG Bo WEI Jun
(Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract Nonfunctional features of distributed component systems usually supported by the crosscutting concerns provided by the middleware nowadays. Although this separation of concerns helps make the system design easy, it enlarges the difficulties of performance prediction at design time. This paper proposed a performance prediction method of distributed component-based systems through aspect templates. This method will automatically weave reusable aspect templates and build a complete performance model based on performance factors of middleware. The results of the model can help designers to find the defects of their designs or make a good choice among different alternative solutions.

Keywords Performance prediction, Middleware, Aspect oriented, Template

1 介绍

性能是软件系统最关键的质量属性之一,受到设计、编码和执行环境等各个因素的影响。但软件工程方法通常着重于关注系统的功能需求,而性能等非功能需求则被推迟到开发阶段后期予以考虑,这种方法可能造成大量的人力投入,额外的昂贵硬件,甚至于设计重构等严重后果。

为解决这个问题,性能预测技术被引进了软件工程设计方法[1]。通过将软件体系结构转化为性能模型,从而在软件设计时进行性能预测,这样可以尽早发现设计中存在的缺陷。

过去几年里,性能预测自动构建技术降低了软件体系结构到相应的性能模型转化的难度。该技术通过在软件体系结构中注入相关的性能信息,可将其自动转化为可求解的性能模型。相关领域已有一些研究成果:有些学者采用扩展 UML 模型[2]进行建模描述,而另一些则进一步研究如何包含中间件性能影响的因素[3,4]。这些方法所建立的软件模型,都可自动转化为分层排队网[5-7]或者随机 Petri 网[8]等性能模型。

尽管这些工作使得自动构造性能模型成为可能,但分布式组件系统通常都是通过中间件平台进行构造的,一些性能因素难以刻画所以仍然难以进行有效的性能预测[9]。比如常见的中间件 EJB 和 CORBA 等,它们为分布式组件系统提供的非功能支持(如并发支持、日志、安全、和事务等服务)通过横切关注点(crosscutting)引入,并且成为分布式组件系统通用的基础服务。它们的性能影响并不被分布式组件系统的设计人员所熟悉,因此需要有一种能自动引入横切关注点性能因素的方法,使其可以完善分布式系统的性能建模,并且可以在应用间复用,以减少性能建模的工作量。

本文主要贡献在于设计了一种基于方面模板的分布式组件系统性能预测方法,即将每个横切关注点作为性能方面构造可复用的性能模板,该模板可在不同的应用中实例化为不同的上下文相关的性能方面模板。中间件的其它性能影响因素则被抽象成数学模型描述的性能模板,性能建模时通过面向方面的建模技术将这些模板实例化并组装成完整的性能模型,从而可以对分布式组件系统进行更为准确的性能预测。

本文第 2 节给出了相关工作比较;第 3 节总体概述了本文采用的方法;第 4 节说明了如何使用面向方面的建模技术建立整体的性能模型;第 5 节分析了中间件自身的性能影响因素;第 6 节给出性能预测实验分析;最后对本文方法小结并

到稿日期:2009-01-16 返修日期:2009-02-20 本文受国家自然科学基金(No. 60573126, 90718033),国家重点基础研究发展计划项目(No. 2002CB312005),国家高技术研究发展计划项目(No. 2006AA01Z180, 2007AA01Z134),国家科技支撑研究发展项目(No. 2006BAH02A01)资助。

黄 翔 博士生,主要研究方向为软件工程等;张文博 博士,助理研究员,主要研究方向为软件工程等;张 波 博士,副研究员,主要研究方向为软件工程等;魏 峻 博士生导师,研究员,主要研究方向为软件工程等。

指出下一步工作方面。

2 相关工作比较

分析中间件对分布式组件系统性能的影响，最简单、最直接的方式是构建包括中间件因素在内的完整软件体系结构的性能模型[5-8]。但这对分布式组件系统的设计人员来说是不太切合实际的，他们往往并不了解中间件的具体实现，因而无法准确描述中间件的性能因素。

包括中间件在内的完整性能模型也可通过组合预定义的中间件性能模板来构造[18,19,23]。但是这些通过数学模型描述的性能模板难以让仅熟悉 UML 模型描述的分布式组件系统的设计人员掌握。

T. Verdickt 提出了一种为 UML 软件模型自动引入中间件性能属性的方法[4]。该方法基于模型驱动架构(Model Driven Architecture)，通过模型转换算法，将中间件无关的体系结构转化为中间件相关的体系结构。作者考虑到了分布式组件系统调用第三方服务的性能影响，但是并没有考虑中间件自动提供的可配置服务的性能影响。因此，使用这种方法分析中间件提供的可配置服务时，要么忽略其影响，要么要求设计人员根据需求将可配置服务修改为第三方服务进行建模。

H. Shen 提出了一种使用面向方面的建模方法[17]，分析给定性能方面对系统整体性能的影响。但是该方法主要用于解决软件通用的性能预测问题，而并未将中间件性能因素考虑在内。本文的方法借鉴了该工作在模板编织方面的思想，但与之不同的是本文的方法重点考虑了中间件横切关注点对分布式组件系统的性能影响，并针对这些横切关注点设计了可复用的性能方面模板。

3 方法概述

3.1 中间件系统中的方面

本文研究目的是设计一种设计时分布式组件系统性能预测的方法，使用一个运行在特定中间件平台上的 EJB3 分布式组件系统[10]实例说明该方法。

在 EJB 组件模型里，拦截器会在组件业务方法调用之前被调用。针对这一特点，使用面向方面建模[12]的方法，通过拦截器实现横切关注点的分离。而其它组件模型，比如 CORBA[13]也存在类似的特性。

在一个组件的业务方法被调用之前，可能会有若干个可配置服务(如安全、日志、事务等)先于它被调用，这些服务都通过横切关注点实现，如图 1 所示。在本文方法中，不同的横切关注点使用 UML 描述的性能模型表示。

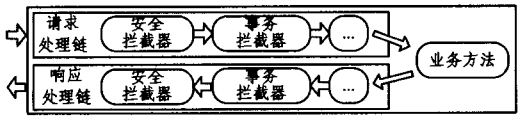

图 1 组件业务方法请求流程

3.2 性能预测方法

为了分离中间件横切关注点对分布式组件系统设计的影响，本文方法采用面向方面的建模方法对其进行体系结构建模，该体系结构模型将包含了一个原始模型和一组面向方面模型。原始模型描述了用户的核心设计，而面向方面模型为原始模型补充了中间件横切关注点的设计[14]。

在本文的性能预测方法中，该体系结构模型需遵循模型驱动架构[15]，通过模型转化算法，可以将软件体系结构模型转化为性能模型。模型驱动架构将包含平台无关模型和平台相关模型，平台无关模型关注于逻辑实现，独立于具体实现技术和支持平台。平台相关模型则全部或部分的给出了底层实现技术和相关支撑平台的实现细节。体系结构模型转化过程如图 2 所示。

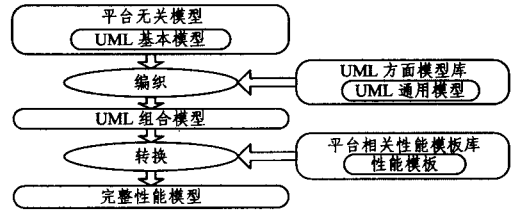

图 2 体系结构模型到性能模型的转化过程

本文性能预测方法的输入是一组遵循 UML 2 规范的平台无关模型，使用标准的 UML 特性文件 SPT[11]标注了性能属性。输出则是一个包含中间件平台性能因素的平台相关模型。在转化过程中会产生一个中间模型，称之为 UML 组合模型。它包含了原始模型和面向方面模型的所有信息，但并不包含底层中间件平台的性能因素。所以，它既可以看作是 UML 原始模型所产出的平台相关模型，也可以看作是生成最终性能模型的平台无关模型。

转换过程中所需要的中间件性能因素分别存放在 UML 方面模型库和平台相关模板库中。UML 方面模型库存放了若干可配置服务的方面模板。平台相关模板库则存放了底层中间件的性能因素的性能模板，本文使用分层排队网模型作为性能模型。

已有研究[5-8]解决了将软件模型转化为性能模型的问题，将不再赘述 UML 组合模型到性能模型的具体转化过程，该转化过程将使用协作图、部署图和活动图等。

4 方面模板

面向方面模型由原始模型和一组方面模型组成。通过编织，将各个方面模型织入原始模型，得到一个包含所有方面的组件模型。本文中，方面模型以 UML 模板形式存放在 UML 方面模型库。

使用一个 EJB3 组件系统实例说明该建模过程。该实例由一个有状态会话 Bean 和一个客户端组成。会话 Bean 使用容器管理的事务策略，由容器自动管理事务，客户端直接对服务器进行访问，共同构成一个典型的“客户/服务器”分布式组件系统。

为了方便比较，设计了两种不同的事务管理器，它们具有相同的架构。不同之处仅在于管理器实例获取上，一种设计只提供单个活动的实例，所有请求都必须竞争该实例的使用权，而另一种设计则为每个请求提供一个可共享的实例，对实例不会产生竞争。本节介绍事务管理器结构，而性能预测结果将在第 6 节阐述。

4.1 原始模型

原始模型包含了如下性能预测必须的元素：

• UML 部署图，描述组件的物理部署状态，以及它们之间的联系方式。

· UML 协作图，描述不同组件间交互的请求模式。

· UML 活动图，描述组件使用场景，并用 SPT 标注必须的性能影响因素[16]。

图 3 描述了本文实例的 UML 原始模型，由两个运行在不同主机上的组件组成，分别为 Client 和 Server 组件，并有一个同步请求发生。

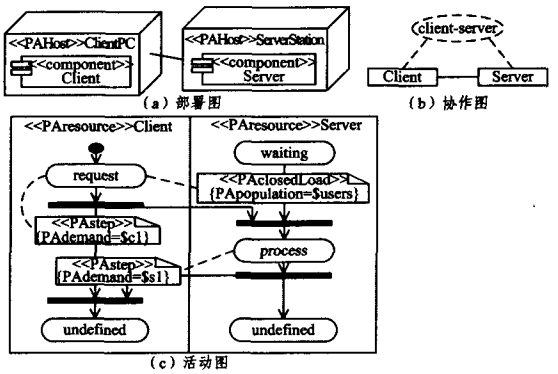

图 3 基本模型

图 3(a)描述了该分布式系统的部署结构。处理器由〈〈PAhost〉〉标注的节点表示，包括了调用策略，以及需要测试的属性(吞吐率、响应时间和处理器利用率等)等信息；图 3(b)说明 Client 和 Server 使用了“客户机/服务器”的请求模式，客户机向服务器发出一个同步请求，并一直保持阻塞，直到服务器发回响应；图 3(c)详细描述了请求活动细节。Client 向 Server 的 process 方法发出一个请求。〈〈PAresource〉〉表示一个软件单元运行在属于自己的线程中。活动则由〈〈PAstep〉〉表示，其中 PAdemand 属性给出了活动占用主机资源的开销。〈〈PAclosedLoad〉〉给出了并发用户数。

不同的 UML 图之间的元素通过命名关联。同一个组件在不同的图中须具有相同的名称。比如说 Client 组件，在部署图里表示了一个组件实例，在协作图里表示为一个分类器，而在活动图里表示为一个泳道。

4.2 通用方面模型

通用方面模型描述了该方面的逻辑结构，但尚未与实际应用相关的属性和配置绑定，通过绑定则可实例化为不同的上下文相关方面模型。上下文相关模型则是部署到特定服务器具有特定配置的软件模型，包含了性能预测所需的实际属性。

本文描述的通用模型用例是一个事务管理的逻辑结构。数据库事务是一组操作，用于数据库管理系统这类需要保证一致性和可靠性的系统。根据数据库的定义，事务必须保证操作的原子性、一致性、隔离性和持久性。由于篇幅限制，将仅描述一个简单的事务管理器的方面模型。该管理器首先会启动一个事务，接着执行相关对数据库操作，之后如果无异常发生则会提交事务，否则回滚事务。

容器管理的事务使用拦截器实现，被部署到与组件相同的服务器上，并使用同步操作将请求转发到目标组件。因此，对于通用方面模型，无须提供协作图和部署图。

图 4 给出了事务管理器的通用方面模型的模板，补充了用户未描述的请求调用部分。图中首字母用“|”标示的属性值会在编织过程中替换为实际属性值。|JoinPoint 是一个连接点，标示了横切关注点将被织入的位置。其它组件对|EndPoint组件的请求，会首先被|TransactionManager 事务管理器组件所拦截。参数 $I 表明事务管理的拷贝个数，用于区别所提供的两种不同的事务管理器设计。当接收到请求，事务管理器会启动事务，并将请求发送到|JoinPoint 方法，然后检查是否有异常发生，最后将结果返回给客户端。为了模拟选择操作，用 PAprob 表明某个路径可能发生的概率。

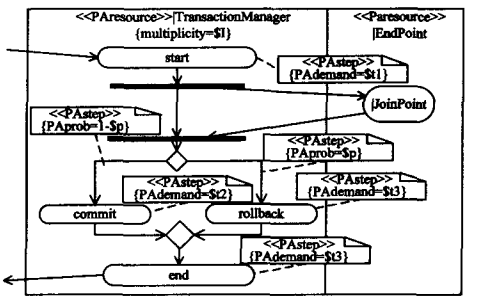

图 4 通用模型

通用模板里的标注信息采用了参数化的方式存放，而不是采用实际值的形式，目的是为实际应用留下可配置的占位空间。参数将会在实例化过程中被赋予与实际应用相关的值。

4.3 上下文相关模型

在通用方面模型编织到原始模型之前，需要为特定的应用上下文绑定不同的规则，将通用模型实例化为特定的上下文相关模型。绑定规则由两部分组成：一是模板绑定，一是参数绑定。

模板绑定的目的是将通用模板转化为上下文相关模板，为通用模板关联资源和分配连接点等。绑定规则可以描述为如下形式的值对“(参数化元素名，实际元素名)”。本文使用用例的绑定规则示例如下：

(|TransactionManager，TransactionManager)；

(|EndPoint，Server)；(|JoinPoint，process)。

参数绑定采用附属资源文件的方式进行，所有参数都会存放在一个指定文件中。比如说图 4 中表示事务启动所消耗处理器资源的参数 $t1，就会被赋予如下值“('assm'，'mean'，($t_start，'ms'))”。这里的 $t_start 是一个分层排队网模型的参数，将在求解前被指定。

4.4 编织

编织是将不同的方面模型织入原始模型，从而构成一个完整的请求模型的过程。需要对 3 种不同类型的 UML 模型分别进行编织。

编织部署图和协作图的过程相对容易，只需要为原始模型添加方面模型中新增的组件。这些新增组件将被分配到相应的主机上，并进一步为其补充与其它组件的协作方式。默认情况下，新增的组件将与目标组件分配到同一台主机上，并采用“客户机/服务器”请求模式与目标组件协作。图 5(a)，(b)给出了编织后的状态。

活动图的编织要将上下文相关方面模型织入到原始模型中。编织的过程需要在原始模型中确定合适的连接点，并将方面模型织入到该连接点，方面模型可能需要针对原始模型做出适当的调整，以满足原始模型占用资源的特点。

图 5(c)给出了活动图编织后的状态。本例中，事务管理器的连接点是 process 方法。织入过程包括选定连接位置，和

插入方面模板两个步骤。从 Client 发出的请求，首先将会被 TransactionManager 拦截，待完成数据库事务相关准备工作后，请求被转发到 Server。经过编织，TransactionManager 和 Server 构成一个新的组件，它们可以再次被作为目标被组合。如果需要，这种结构可以一直迭代下去，直到所有必需的方面都被织入目标组件。图 6 给出了一个具有安全检查和事务管理的多方面编织实例。

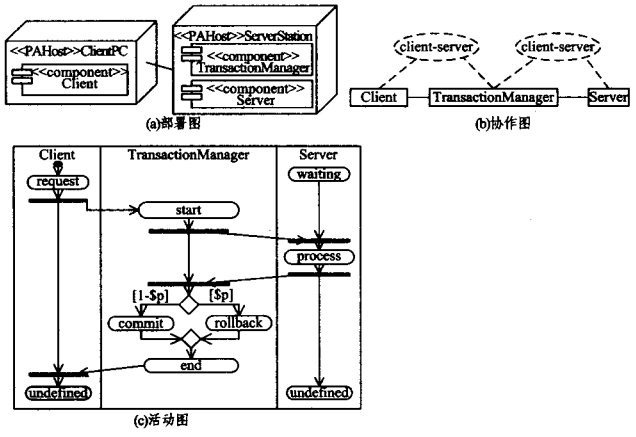

图 5　编织后的系统模型

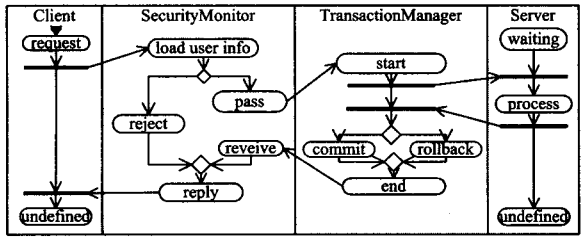

图 6　多拦截器编织实例

为了让我们的方式更简单，更具有适应性，对常规编织和建模方法[14,17]做出了相应简化，只需考虑调用链的编织，而不需考虑组件的静态结构等情况，所需支持的连接点也只有方法调用一种。

5　构造性能模型

除了横切关注点会对性能产生影响外，底层的中间件平台也会对性能产生影响。为了将这种因素也考虑进来，为不同类型的组件建立了与之对应的性能模型。因为对于同一类组件，底层环境的处理是一致的，并且这种容器底层的操作无须应用设计人员参与。所以，对于中间件的影响因素，直接使用数学模型进行描述。

对底层中间件影响因素建模的方法，借用了其它基于模板方法[18,19,23]的思想。同时，考虑到方面编织时存在的嵌套性，对生成的方面性能模板做出相应调整，为方面模型留出占位空间，使之可以适应嵌套结构的存在。并在进行模型转换时做必要的分割，找出编织后组件的边界，以便与性能模板进行组合。

5.1　中间件性能模板

采用分层排队网模型作为分析性能的数学模型，所讨论的性能模型也基于此数学模型，但并不局限于该模型。本节将以有状态会话 Bean 为例，说明性能模板的构造以及与 UML 模型的组合。

一个请求在到达组件之前，中间件平台自身需要做大量的工作，诸如网络通信、数据转换、资源分配和服务查找等操作。这些操作都会消耗一定的处理器时间，并对最终系统的性能造成影响。所以必须将中间件的影响因素考虑进来。下面简化了中间件平台整个调用的流程，仅列出了几个关键点，而将其它详细信息合并进这些关键位置。

图 7 给出了有状态会话 Bean 的性能模板。中间件服务被设计为两个任务：一个是 Container 任务，具有无限实例个数，表示该对象可以在不同请求间并发执行，不会受到同步操作的影响；另一个是 ContServ 任务，只有一个实例，是关键区域竞争的抽象，模拟同步处理操作。另外 Bean thread Pool 模拟了实例池的大小，由参数 $M 控制池的规模。Bean 实例的挂起与激活操作则由 CallBack 任务进行模拟，参数 $p_s 用于申明挂起与激活的概率。方面子模型（Aspect Submodel）是为 UML 方面模型预留的展位空间。各任务对处理器资源的消耗以首字母为“$”的参数表示。这样，同一个模板便可以适用于不同的硬件环境。

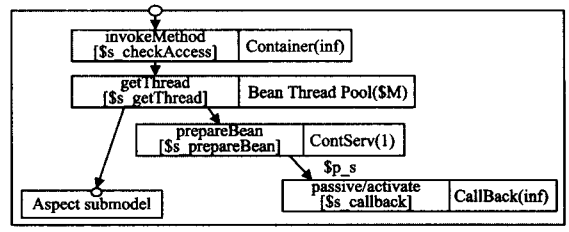

图 7　性能模板以及嵌套结构

性能模板中的小圆圈是该模板的输入接口，其它模板通过这个接口与该模板组合。方面子模型为一个占位空间（Aspect submodel），仅保留了一个接口位置，实际模板则根据实际需要由与之关联的 UML 模型转换而来。模型转换与组合细节在下一节给出。

5.2　模型转换与组合

获取一个完整的性能模型的步骤如下：首先，分析组件类型，根据运行环境以及组件类型选择合适的底层中间件性能模板；之后，将由 UML 模型转换而来的性能模板与底层中间件性能模板进行组合；最后，分析不同组件间的协作方式，为不同组件间建立相应的联系。

使用图转换算法[6,7]，UML 方面模型可转化为分层排队网的性能模型。图 8 给出了完整的性能模型，其中的方面子模型（Aspect Submodel）由转换算法转换而来。转换过程需要划分出 Client 与 Server 组件间的边界，以便于将中间件性能模板插入其间，组合生成最终性能模板。子模型由两个任务组成：一个是 Server 任务，由会话 Bean 组件转化而来；另一个则是由事务管理器转化而来的 TransactionManager 任务，其中的矩形与活动图中的活动对应，虚线箭头代表消息返回，实线箭头代表同步调用，整个结构与 UML 模型等价。

由于 Server 组件是一个有状态会话 Bean 组件，因此在组合过程中选用了如图 7(a)所示的中间件性能模板，与方面子模型组合一同分配到 ServerStation 上。Client 组件则转换为一个引用任务，模拟多用户请求，参数 $N 代表并发用户数，参数 $think_time 模拟用户思考下一步操作的时间间隔。Client 与中间件平台，中间件平台与组件间，均通过同步请求方式协作。

图 8 描述了组合后的完整性能模型。请求首先需要经过

容器的相关处理，然后转发到相应的横切关注点，最后才会被转发到目标组件。

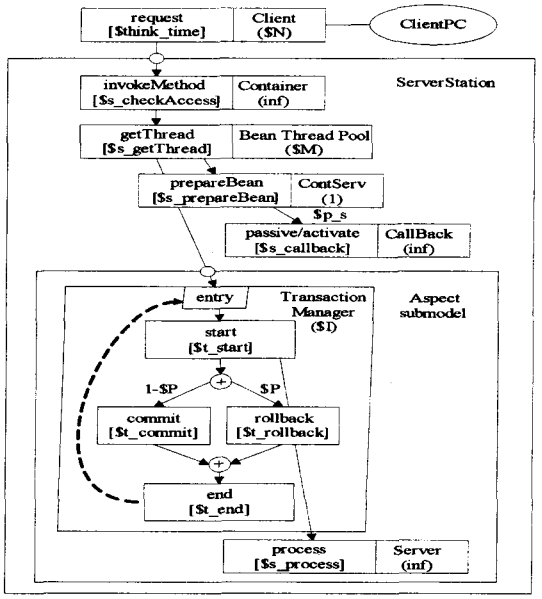

图 8　完整性能模型

6　实验验证

本节通过实验验证方法的有效性。采用的性能模型是分层排队网模型，可以通过分析工具 LQNS 和模拟工具 LQN-Sim 进行求解[21]，结果包含吞吐率(请求/秒)，处理器利用率(处理器占用百分比)等数据。

6.1　测试环境

测试环境由一台客户机和一台服务器组成。通过 100Mbps 以太网连接。客户机配置为奔腾 IV 1.5GHZ 和 512MB RAM。服务器配置是奔腾 IV 2.0GHZ 和 1024MB RAM。两台机器均使用 RedHat 9 操作系统。

EJB 容器选用 OnceAs v3.0[22]，运行在 JDK1.5 上。客户端直接通过应用程序请求服务器，而不经过 HTTP 服务器。由于篇幅问题，没有显式刻画出数据库因素，但是使用 MySQL V5.0 作为数据库服务器，测试中数据库未成为应用的瓶颈。

测试用例中用户思考时间被设为 0，一个请求结束时会立即重新发起请求，使系统任何时刻都保有固定数量的并发用户请求。分层排队网模型所需的参数保存在一个独立的配置文件中，数据可以根据不同的软硬件平台进行切换。

6.2　预测结果

作为对比，给出了两个不同事务管理器方案的预测结果。一种方案提供了集中式的管理策略，所有组件都需要竞争获取到该管理器的活动对象后才能执行。对于这种设计，参数 $I 设为了 1，称它为独享管理器。另外一种方案则是在不同请求间共享同一个实例，参数 $I 被设为无限(infinite)，称它为共享管理器。由于篇幅问题，并没有显式地给出数据库竞争，而是将开销全部放在组件的总开销上。除了上述不同，这两种事务管理器都拥有相同的结构，如图 8 所示，对单一请求具有相同的相应时间。

虽然可以仅通过改变参数 $I 达到预测不同设计的效果，但是为了演示方法在方面切换方面的简便与快捷性，仍然为不同的设计构造了不同的通用方面模板。为了测试这两种不同的方案，只需在测试前申明选择哪种模板，本方法会自动地完成所有余下工作。

在测试环境下提取出了如下的参数：

$s_process=0.175ms；$s_checkAccess=1.325ms；

$s_getThread=0.0135ms；$s_prepareBean=0.472ms；

$s_callback=0.513；$t_start=0.006ms；

$t_commit=0.5205ms；$t_roolback=0.566ms；

$p=0.05；$p_s=0.02；

$t_end=0.0025ms；$M=10。

图 9 给出了预测结果的比较。图 9(a)给出了平均吞吐率的预测结果。可以看出对于共享管理器在 10 个客户到达时系统达到饱和状态。而对于独享管理器 5 个用户则到达饱和，并且随着用户量的增加，吞吐率会立即存在下降的趋势。

图 9(b)给出了服务器处理器利用率的预测结果。可以看出对于共享管理器，吞吐率达到饱和时，处理器总利用率也接近 100%。而对于独享管理器，处理器利用率不到 80%。处理器利用率不足是由竞争唯一的处理器对象引起的，因为独享管理器的利用率明显低于共享管理器的利用率，而且随着用户数的增加，利用率会有所下降，更多的处理器时间花销在资源竞争上。设计人员根据预测结果可以在不同的设计间作出正确的抉择，并可辅助他们发现引起性能问题的潜在原因。

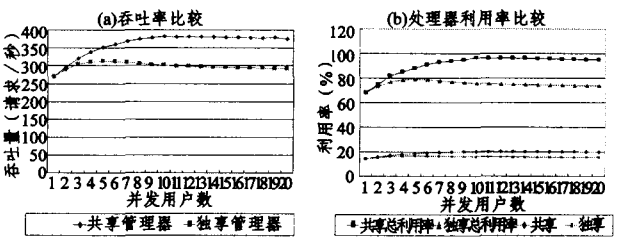

图 9　预测结果分析

虽然实际的方面模型可能会更复杂，但是分析的方法一样。通过切换和组合不同的备选方案，即可构造出相应的完整性能模型。

性能预测结果可以帮助设计人员对不同负载压力下的性能问题有一个比较清晰的认识，并辅助他们尽早发现设计中的缺陷，或者从多个可选方案中筛选出一个相对最优的方案。

结束语　本文研究了一种基于方面模板的分布式组件系统性能预测方法，可以自动地为横切关注点进行性能建模，并分析它们对系统总体性能的影响。不仅考虑了不同横切关注点所带来的性能影响，而且包含了底层中间件系统所带来的性能因素。

横切关注点被设计为可复用的方面模板，该模板在实例化为与特定应用相关的模板之后，将被编织到组件的原始模型之中，进一步通过与底层中间件性能影响因素模板的组合，构成一个完成的性能模型，并可以直接生成性能预测结果。

系统设计人员可以通过对多个候选方案的切换，或者对多个所需方面的组合，构造一个完整的性能模型。预测结果则能辅助设计人员做出更好的抉择，发现潜在的性能问题，以降低系统开发后期的开发和维护费用。

下一步，计划进一步优化和抽象其它目前尚未考虑的性

(下转第 157 页)

Web服务的描述语言到模型检测所需的形式模型的转换方法，并考虑采用组合验证和符号化算法优化模型检测性能，从而推动Web服务形式化验证的实用化。

参考文献

[1] Curbera F, Goland Y, Klein J, et al. Business Process Execution Language for Web Services[OL]. ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf, 2002

[2] Pistore M, Traverso P, Bertoli P. Automated Composition of Web Services by Planning in Asynchronous Domains[C]// Proc. ICAPS'05. 2005

[3] Fu X, Bultan T, Su J. Analysis of interacting BPEL web services [C]// WWW'04. ACM Press: 621-630

[4] Huang Hai, Tsai Wei-Tek, Paul R, et al. Automated model checking and testing for composite web services[C]// ISORC'05. IEEE Computer Society: 300-307

[5] Halpern J, Vardi M Y. Model checking vs. theorem proving: A manifesto[R]. IBM Almaden Research Center, 1991. An extended version of a paper in Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning, 1991

[6] van der Meyden R. Common knowledge and update in finite environments. I: extended abstract[C]// Proc. of the Conf. on Theoretical Aspects of Reasoning about Knowledge. 1994: 225-242

[7] van der Meyden R, Shilov N S. Model checking knowledge and time in systems with perfect recall[C]// Proc. Conf. on Software Technology and The oretical Computer Science, LNCS No 1738. Berlin: Springer, 1999: 262-273

[8] Benerecetti M, Giunchiglia F, Serafini L. A model checking algorithm for multi-agent systems[C]// J. P. Muller, M. P. Singh, A. S. Rao, eds. Intelligent Agents V, volume LNAI Vol. 1555. Berlin: Springer-Verlag, 1999

[9] van der Meyden R, Su Kaile. Symbolic model checking the knowledge of the dining cryptographers[C]// Proc of 17th IEEE Computer Security Foundations Workshop. June 2004: 280-291

[10] van der Hoek W, Wooldridge M. Model checking knowledge and time[C]// Proc. 19th Workshop on SPIN (Model Checking Software). Grenoble, April 2002

[11] Su Kaile, Sattar A, Luo Xiangyu. Model Checking Temporal Logics of Knowledge Via OBDDs[J]. The Computer Journal, 2007, 50(4): 403-420

[12] Fu Xiang, Bultan T, Su Jianwen. Model checking XML manipulating software[C]// Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2004. Boston, Massachusetts, USA, July 2004: 252-262

(上接第125页)

能影响因素，比如说数据库连接池和线程池的调度等。

参考文献

[1] Smith C U, Williams L G. Performance Solutions[M]. Addison-Wesley, 2002

[2] Cortellessa V, Mirandola R. PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams[J]. Science of Computer Programming, Elsevier Science, 2002, 44 (1): 101-129

[3] Petriu D B, Woodside M. Software Performance Models from System Scenarios in Use Case Maps[C]// Proceedings of International Conference on Modeling Techniques and Tools for Performance Evaluation. LNCS 2324. 2002: 141-158

[4] Verdickt T, Dhoedt B. Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models [J]. IEEE Transactions on Software Engineering, 2005, 31(8): 695-711

[5] Petriu D C, Shen H. Applying UML Perfromance Profile: Graph Grammar-based Derivation of LQN Models form UML Specifications[C]// Proc. of International Conference on Modelling Techniques and Tools for Performance Evaluation. LNCS 2324. 2002: 159-177

[6] Woodside M, Petriu D C. Performance by Unified Model Analysis (PUMA)[C]// Proc. 5th Int. Workshop on Software and Performance. Palma de Mallora. July 2005: 1-12

[7] Woodside M. From Annotated Softwae Designs (UML SPT / MARTE) to Model Formalisms[C]// M. Bernardo and J. Hillston, eds. SFM. LNCS 4486. 2007: 429-467

[8] Lopez-Grao J P, Merseguer J, Campos J. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering[C]// ACM Proc. of Workshop on Software and Performance. 2004: 25036

[9] Woodside M, Frank G. The Future of Software Performance Engineering[C]// IEEE Future of Software Engineering (FOSE' 07). 2007: 171-187

[10] Jendrock E, Ball J, Carson D, et al. The Java EE 5 Tutorial [OL]. http://java.sun.com/javaee/5/docs/tutorial/doc/, Sun MicoSystems

[11] Object Management Group. UML Profile for Schedulability, Perfromance, and Time. 2005

[12] Roman E, Sriganesh R P. Mastering ejb3 or specification[M]. Addison-Wesley, 2006

[13] Object Management Group. The Common Object Request Broker: Architecture and Specification. 2002

[14] France R, Ray I. An Aspect-Oriented Approach to Early Design Modeling[C]// IEE Procceedings-software, Special Issue on Early Aspects: Aspect-oriented Requirements Engineering and Architecture Design. 2004: 173-185

[15] Miller J, Mukerji J. MDA Guide[M]. June 2003

[16] Petriu D C, Woodside M. Performance Analysis with UML[M] // B. Selic, L. Lavagno, G. Martin. UML for Real, 2003: 221-240

[17] Shen H, Petriu D C. Performance Analysis of UML Models using Aspect Oriented Modeling Techniques[C]// Model Driven Engineering Languages and Systems. LNCS 3713. 2005: 156-170

[18] Xu J, Woodside M. Template-Driven Performance Modeling of Enterprise Java Beans[C]// Proc. Workshop on Middleware for Web Services. Enschede Netherlands, 2005: 57-64

[19] Xu J, Oufimtsev A. Performance Modeling and Prediction of Enterprise JavaBeans with Layered Queuing Network Templates [C]// ACM Proceedings of Workshop on Specification and Verification of Component-Based Systems. 2005

[20] Franks R G. Performance Analysis of Distributed Server Systems[C]// Department of Systems and Computer Engineering. Carleton University, Ottawa, Ontario, Canada, December 1999

[21] Woodside M, Franks G. Tutorial Introduction to Layered Modeling of Software Perfromance[OL]. http://www.sce.carleton.ca/rads/lqns/lqn-documentation

[22] http://www.once.com.cn

[23] Wu X, Woodside. Performance Modeling from Software Components[C]// ACM Proc. of Workshop on Software and Performance. 2004: 290-301