资源敏感的分布式系统性能建模方法

黄 翔[1,2] 陈志刚[1]
(中国能源建设集团广东省电力设计研究院 广州 510663)[1]
(中山大学信息科学与技术学院 广州 510006)[2]

摘 要 早期的性能预测可帮助设计人员有效地评估和改进系统设计,降低性能风险,提高软件制品满足性能需求的可信程度。但复杂低效的性能模型构造过程,严重阻碍了它在软件开发过程中的应用。为简化建模复杂度,以典型的UML模型为基础,研究了一种面向分布式系统的性能模型方法,并提出了一种中间模型——资源场景模型(Resource Scenario Model,RSM),解决了多UML视图转化为多性能模型的问题,使设计人员可以根据自身偏好选择熟悉的工具进行性能建模和性能分析。最后,以UML活动图和序列图到分层排队网和通用随机Petri网模型的转换为例,说明了本方法的可行性和有效性。

关键词 资源,分布式系统,性能模型,模型转换
中图法分类号 TP32 **文献标识码** A

Performance Modeling Approach for Resource Sensitive Distributed Systems

HUANG Xiang[1,2] CHEN Zhi-gang[1]
(Guangdong Electric Power Design Institute, China Energy Engineering Group Co. Ltd., Guangzhou 510663, China)[1]
(School of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510006, China)[2]

Abstract Performance prediction of software at early stage makes benefit to improve the quality of designs and reduce the performance problems. However, the huge costs of performance modeling approaches make it impossible to integrate the approach into the software development processes. In order to develop a cost effective modeling approach, we used the widely accepted UML model to support our work for distributed systems, and designed an intermediate model named Resource Scenario Model(RSM) for multi architecture model to multi performance model transformations. Therefore, designers can use their preferred tools to build and analyze software's performance. To illustrate the effectiveness of our method, we given a case study, in which UML models can be easily transformed into performance models, such as LQN and LGSPN.

Keywords Resource, Distributed system, Performance model, Model transformation

1 引言

分布式系统,已经成为软件系统的一种主要体系结构模式。由于这类系统固有的复杂性和高并发性,通常会对性能提出一定的要求。然而,已有方法须等到系统实现后才能进行性能测试,因而往往需为此付出昂贵的调优或重构代价。早期的性能分析,即基于性能模型的分析方法,可通过模型发现软件系统存在的性能缺陷,从而大大降低软件开发成本,提高软件制品满足性能需求的可信程度[1]。

性能模型一般采用某种数学方法刻画一定场景下系统处理用户请求的过程,包括软件间的交互以及对硬件资源的占用等。然而,对软件开发人员而言,数学模型的学习曲线异常陡峭,严重阻碍了性能预测的开展。因而,为软件开发人员提供一种便捷的建模方法,降低性能建模的门槛,成为性能建模领域广泛关注的重点与难点[2,28]。

UML模型[3]作为一种成熟的体系结构描述语言,已被设计人员广泛接受和使用。OMG组织提出的SPT[4]性能标注规范,为UML性能建模提供了基础支持。SPT的基本思想是通过为UML模型中的元素添加性能标注(如处理器占用时间和调度策略等),丰富其在性能属性方面的描述能力。

然而,SPT并未增强UML模型描述软件占用硬件资源的描述能力。比如,传统的体系结构模型(比如UML模型)侧重于功能描述,即组件之间的交互关系等,但在分布式系统里,影响性能的关键往往是对核心资源的使用(如线程池、连接池等)。对资源的获取与释放的过程,不同于组件的调用关系,涉及到资源量、使用量和使用方式等多种非交互性的特征。

此外,用户通常希望使用自己熟悉的模型进行建模和分析[14]。但在实际应用中,不同的设计人员往往拥有不同的偏好,模型的选择也与知识背景和经验密切相关。比如UML模型可以选择活动图和序列图,甚至同一视图的不同版本,另一方面性能模型也同样具有多样性,如排队网模型和Petri网模型等。若要在不同的体系结构模型和性能模型之间进行转换,则会面临 $M*N$ 的转换问题,转换代价非常大。

到稿日期:2012-11-19 返修日期:2013-03-27 本文受国家自然科学基金(61272013)资助。
黄 翔 博士后,主要研究方向为软件工程等;陈志刚 高级工程师,主要研究方向为系统规划等。

针对上述问题，本文通过分析资源获取与释放的特点，扩展了 SPT 描述资源使用的方法，并进一步以资源为中心，设计了体现分布式系统资源使用的中间模型，使其可以成为模型转换的中介，将多模型间转换的复杂度降为 $M+N$。

本文第 2 节给出了相关工作比较；第 3 节介绍了 RSM 模型构造的基础，即资源与资源依赖关系；第 4 节给出了性能建模方法和建模实例；第 5 节则描述了模型转化方法和实例转换结果；最后对本文进行总结。

2 相关工作比较

软件开发过程早期的非功能属性评价是近年来非常活跃的一个研究领域。相关工作中，基于 UML 模型的定量分析被众多方法所采纳。比如在文献[5]中，作者根据带性能标注的活动图以及 WSDL 描述信息，给出了一种评价面向服务的质量协商方法。在文献[6]中，作者设计了一种通过综合活动图以及协作图信息生成离散模拟模型的方法。在文献[7]中，作者提出了一种将活动图转化为随机进程代数的方法。

而在性能工程领域，文献[2]综述调研了大量主流性能模型，并做了全面的分析、比较和总结。其中 Petri 网和排队网是两种使用最为广泛的性能模型，因而本文选用分层排队网和通用随机 Petri 网模型来说明转换方法。对于这两类模型，目前已存在一些专有的转换方法。如将 UML 模型转化为 Petri 网的方法[8-10]，将 UML 模型转化为分层排队网的方法[11,12]等。此外，也出现将其它类型模型转化为性能模型的研究，比如将 PCM 模型转化为分层排队网模型[13]。

虽然，上述方法存在或多或少的差异，但本质上都是将一种特定的模型转化为另外一种特定的模型，且均未充分考虑由于建模侧重点发生变化而造成的建模上的改变和多对多的模型转换问题。本文则着重关注 UML 模型如何适应性能建模需求和多对多模型转换，以提高早期性能预测的可用性。

在文献[14,15]中，作者提出了一种核心场景模型(Core Scenario Model，CSM)。这种模型以场景作为主要研究对象，建立了一种客观反映资源使用情况的模型。该方法已经开始考虑资源，但突出的程度不够，仅分析了资源获取和释放关系，并未考虑复杂系统，比如分布式这类复杂系统对资源重用性的改变。其次，本文在资源获取和释放的基础上，参考体系结构模型与性能模型特点，充分考虑了资源间的依赖关系，并以此为基础，设计了一种资源场景模型(RSM)，它将复杂的图结构分解为多棵具有明确语义的树形结构，以提高组件内部活动的精确性，降低组件之间的依赖性，使模型可以灵活地与其它性能模型进行转换。

在已有的研究工作中，我们沿用传统的建模和转化方法，研究了基于中间件应用系统的性能分析与优化方法[16-18]。而本文针对上述发现的问题，对性能建模和模型转换进行了扩展，提出了以资源为核心的建模与转换方法，并构造了相应的原型系统[19]。

3 系统资源与性能模型

在性能分析领域模型[20,21]中最重要的 3 个要素是用户负载、软件行为和系统资源。它们与性能密切相关，即系统所拥有的资源和软件的执行行为，决定了在一定用户负载下的性能表现。由于 SPT 给予用户负载和软件行为足够的支持，因此本文着重关注系统资源。

3.1 系统资源

资源是系统的重要组成部分，它为软件组件提供服务或运行环境。性能的一个决定因素在于资源处理能力的有限性，当资源处理能力不足时便会造成系统的等待或延迟。

不同的性能模型对资源采用不同的抽象形式。一些模型会严格区分硬件和逻辑资源，如分层排队网；而另一些可能并没有严格的界限，如 Petri 网。资源按其竞争对象实体的不同[20,21]，可分为物理资源和软件资源两类：

- 物理资源：支持软件运行的硬件设备，为逻辑资源提供运行环境，实际调度执行。
- 逻辑资源：即软件组件(如无特别说明，后文中组件专指逻辑资源)，功能的逻辑载体，主要刻画数量和占有状态，如线程池、连接池和无状态功能模块等。

3.2 资源依赖关系

软件执行步骤可分为组件内部的执行流程和组件与组件之间的交互过程。组件内部流程影响组件自身对于资源的使用情况，而交互行为则决定了资源与资源之间的依赖关系。这种依赖在高负载情况下会造成大量的竞争，导致关键资源的拥塞并最终影响系统性能。因此，对于软件行为，最需要关注的是组件间的依赖关系。

按资源获取与释放关系的不同[20,21]，资源依赖可分为嵌套和非嵌套依赖两种。

- 嵌套依赖指资源的获取与释放的顺序匹配，典型的如同步调用。如果放宽释放的限制，对于不必显式释放资源的情况，只关注于资源获取的顺序，则异步调用也可归于此类。
- 非嵌套依赖指资源的获取与释放的顺序不匹配，也就是说资源的获取步骤与释放步骤在不同的组件中。典型的如池资源，一个组件获取该资源，后续某个组件才释放该资源。

4 性能建模

UML 模型是一种被广泛使用的体系结构建模语言。它以符号方式，从多个视角，抽象地描述系统架构。使用 UML 进行性能建模，可以降低软件设计人员学习掌握性能模型的难度，促进性能预测过程在软件开发过程中的运用。

为满足性能建模对于资源和行为两方面的需要，在众多 UML 模型视图中，本文选用部署图和活动图这两种视图。另外，由于部分 UML 视图之间存在相似性，活动图也可由序列图替代。但因为活动图能更直接地反映并发、选择、循环等操作，所以本文以活动图作为主要的说明对象。

部署图用于描述物理资源的处理能力和软件部署情况，而活动图则用于刻画软件在一定场景下的行为，即资源之间的依赖关系。SPT 标注则赋予了 UML 模型元素标注性能属性的能力，如资源数量、调度策略、执行时间和负载类型等。但其对资源之间的依赖关系的支持不够，下文将着重分析依赖关系的构建方法。

4.1 嵌套依赖关系建模

对于功能建模，组件之间的交互体现的是功能的调用或数据的流转，不必关心资源的获取与释放。而对于以资源为核心的建模，虽然也在一定程度上体现功能，但更关注于资源之间的依赖关系，也就是需要明确资源的获取与释放时机。

嵌套依赖最大的特点是资源的获取与释放顺序的匹配。因此，可通过“请求与返回”消息进行嵌套依赖的建模。在传统的功能建模中，并不严格要求这一点。图 1 给出了几类基本的分布式系统交互模式，通过它们可组合出各种复杂的场景。

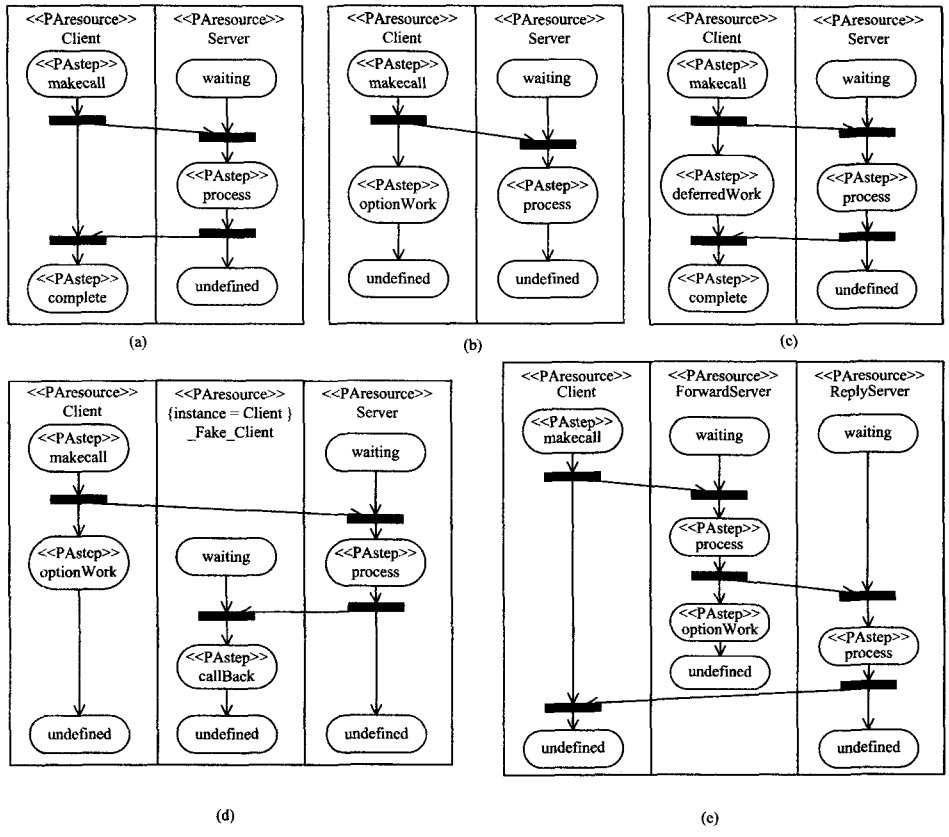

图 1　嵌套依赖关系建模

同步模式(见图 1(a))和异步模式(见图 1(b))是最基本的两种交互模式。Client 到 Server 的请求边代表资源的获取,返回边则代表了资源的释放,异步请求不需要返回边释放资源。

延迟同步(见图 1(c))和异步回调(见图 1(d))是上述两种模型的组合和扩展。延迟同步在等待依赖资源的同时进行部分操作。而异步回调则可看作两次异步调用的组合,其客户组件被分成了两个组件,用于区分请求与回调的角色不同,4.3 节将对其进行详细说明。

请求转发(见图 1(e))比较特殊,多见于代理服务器,当前在中间件技术里受到重视的分阶段请求也适用于这种模式,可看作同步和异步的一种混合组合模式。

图中资源和程序步骤使用了 SPT 进行标注,详细信息可参见 SPT 规范的说明。简单地说,<PAresource>代表逻辑资源、占用数量、调度策略和占用时间[27]等属性;<PAstep>代表执行步骤,具有执行时间、执行概率等属性。另外,部署图由于与传统建模方式并无太大差异,因此没有特别给出。仅需要注意的是物理资源使用<PAHost>标注。

4.2　非嵌套依赖关系建模

非嵌套依赖有助于提高系统性能。典型地,资源将延迟到使用时获取,使用结束立即释放,以缩短占用时间,提高系统性能。

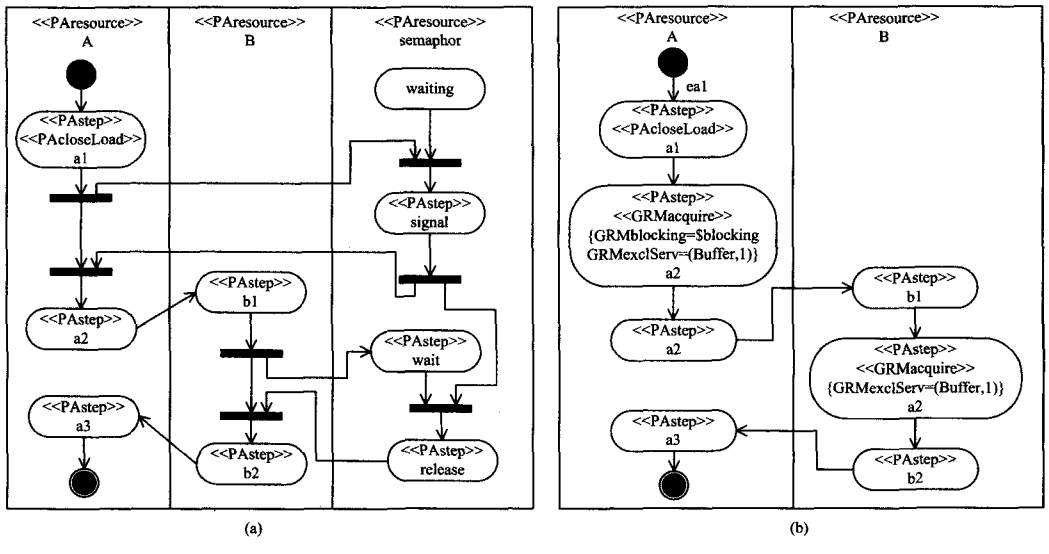

图 2　非嵌套依赖关系建模

嵌套依赖可由调用关系描述,非嵌套则相对复杂,需要支持两个功能:一个获取资源,一个释放资源。获取得到的资源

将一直维持，直到其它组件释放该资源（见图 2(a)）。本文采用 SPT 标注的＜GRMacquire＞和＜GRMrelease＞标签分别刻画资源的获取与释放时机（见图 2(b)）。

非嵌套依赖可以分为阻塞和非阻塞两类。阻塞说明资源的获取采取同步方式，只有在获得资源之后才能继续执行，而非阻塞意味着采用异步方式获取资源，不必等到获得资源后才继续执行。这一区别可通过＜GRMacquire＞标签的阻塞属性（blocking）进行区分。Blocking 属性为一个布尔类型变量。

4.3 建模规约

上述两节介绍了资源之间的依赖关系。依赖意味着潜在地存在资源竞争，即需要保证竞争的正确性和一致性。因此，建模的过程需要遵循一定的规则和约定。

首先，性能建模可以根据需求采用不同抽象程度，但必须保证资源的唯一性；其次，由于组件可能有多个功能入口，组件内部执行步骤的连通性也是必须考虑的因素。

唯一性：在同一个 UML 模型中，任一资源有且只有一个实例。

唯一性保证模型中的资源与实际资源一致。对于回调函数，为了区别不同的功能入口，必须构造一个代理资源。同时为了确保唯一性，本文扩展了 SPT 能力，为＜PAresource＞标签提供了 instance 属性，用该属性将分离的资源关联起来，如图 1(d)异步回调所示。

扩展的代理资源可以继承或覆盖原有资源的所有属性。在资源自身发生嵌套依赖时，可通过覆盖资源的数量（Pacapacity），模拟重入策略。对于复用已分配资源的情况，将代理资源数量属性设为无限（infinite），则能进行模型资源复用；而对于需要重复分配资源的情况，继承原有属性即可。

连通性：连通子图内的执行步骤必须处于同一功能区域（比如一个函数）之内。

连通性指组件内部执行步骤的连通性，而不是组件之间的连通性，详见 5.2 节说明。

5 模型转换

UML 建模为设计人员提供了一条简便快捷的性能建模方法，使性能建模过程与系统设计过程更容易整合。但是性能建模的目的在于尽早获得性能预测结果，UML 模型并不提供这种能力。所以需要将 UML 模型转化为一种可求解的性能模型。

实际应用中，模型转换会面临模型多样性的问题[24]。UML 模型存在多种视图，性能模型也存在多样性。如果将建模语言范围扩展到更宽泛的体系结构描述语言范畴，问题将更加复杂。不同的设计人员拥有不同的设计使用偏好，而不同的性能模型产生的结果又会存在很大的差异。为使设计人员可以根据已有经验，适应性地预测系统性能，就需要解决这种多样性问题。

本文提出了一种基于资源和场景的中间模型（RSM）。场景为一定条件下软件系统表现的行为，而行为则由组件之间的依赖关系构成。该模型不仅保留了 SPT 标注所具有的核心信息，而且直观地反映了性能模型所关注的资源竞争。

5.1 资源场景模型

RSM 模型不仅包含了 SPT 标注里与性能建模相关的核心数据，而且包含了需要从 UML 模型和标注中推导出的信息。图 3 给出了 RSM 的元模型。

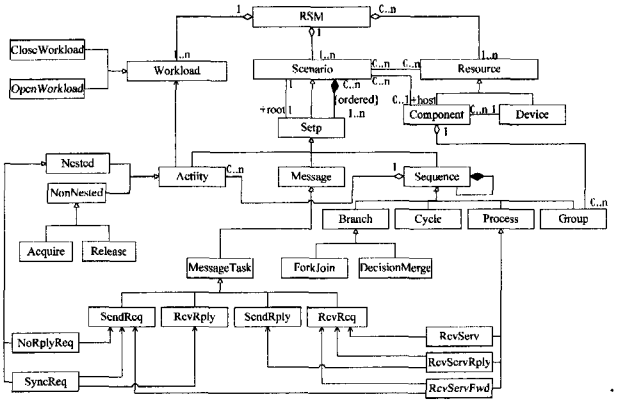

图 3 RSM 模型元模型

RSM 模型包括 3 个主要类型：负载（Workload）、场景（Scenario）和资源（Resource）。它们与 SPT 中的性能领域模型对应，其中最大的不同就在于场景的组织。

直接从 UML 模型转化过来的元素只有活动（Activity）。因为它是一个原子的执行步骤，所以 RSM 模型中保留了这种类型。

执行步骤（Step）在 UML 模型中通过控制流连接。但 RSM 模型中并没有类似的流元素存在，而是用容器类型刻画执行步骤间的联系，包括顺序（Sequence）、并发（ForkJoin）、选择（DecisionMerge）和循环（Cycle）4 种类型。此外，功能区域（相当于一个函数或者服务）对应一个组容器（Group）。

这种设计可以带来两个好处：一是格式规范，便于格式检查；二是具有明确的语义信息，减少了转化为性能模型时对路径的分析。UML 模型中，组件之间的调用（即嵌套依赖）是与性能最紧密相关的信息。RSM 模型将这种依赖关系单独提取出来进行建模，使得依赖关系更加明确，便于转换的实施。

嵌套依赖包含两类主要类型：嵌套活动（Nested）和处理序列（Process）。嵌套活动描述客户组件对于服务组件的请求，包含同步（SyncReq）和异步（NoRplyReq）两类。处理序列则描述服务组件接受客户组件请求后处理并返回的情况，包含 3 类：接受请求并处理（RcvServ）、接受请求处理并返回（RcvServRply）和接受请求处理并转发（RcvServFwd）。通过它们之间的组合，便可构成所有的嵌套建模模式。表 1 给出了这种组合关系。

表 1 RSM 模型依赖关系组合列表

嵌套依赖关系	RSM 组合
同步	SyncReq＋RcvServRply
异步	NoRplyReq＋RcvServ
请求转发	SyncReq＋RcvServFwd＋RcvServRply

异步回调和请求转发是同步和异步模式的组合，因此没有特别给出它们的组合情况。另外，模型中分离了依赖关系，所以嵌套活动和处理序列需要有标识其关联的键，即消息类型（Message）。它包含 4 个子类型：发出请求（SendReq）、接受请求（RcvReq）、发出响应（SendRply）、接受响应（RcvRply）。

最后，非嵌套依赖包含两个用于标识获取（Acquire）和释放（Release）资源的原语类型。

5.2 RSM 模型转换

RSM 模型以资源和资源的依赖关系为核心，UML 到 RSM 模型的转换，主要关注如何推导出资源的依赖关系。本文所提出的转换算法分为 4 大步：连通性分析、依赖关系分离、控制流分解和转换。

5.2.1 连通性分析

连通性分析是本文模型转换方法的基础，用于分析组件内部活动之间的关联性，区分组件内部控制流关联的步骤（直接关联）和组件之间依赖所关联的步骤（间接关联）。

图 $G=(V,E)$ 表示 UML 模型，V 代表活动节点集合，E 代表控制流的集合。V 具有属性集合 $A=\{type, owner\}$，$\lambda_v^{attribute}$ 获取指定活动 v 的属性。边 $e=(u,v)$ 为由起点 u 到终点 v 的边。A 集合中的属性 $type$ 代表了 UML 活动图节点类型，$owner$ 代表节点所在的组件。

直接关联：如果存在边 $e=(u,v)$，且 $\lambda_u^{owner}=\lambda_v^{owner}$，则 u 为 v 的前驱节点，v 为 u 的后继节点，记作 $u\rightarrow v$。

间接关联：如果存在边 $e0=(u0,v0)$ 和 $e1=(u1,v1)$ 为同一请求的请求响应消息，节点满足 $\lambda_{u0}^{owner}\neq\lambda_{v0}^{owner}$ 且 $\lambda_{u1}^{owner}\neq\lambda_{v1}^{owner}$ 且 $\lambda_{u0}^{owner}=\lambda_{v1}^{owner}$，则 $u0$ 为 $v1$ 的前置节点，$v1$ 为 $u0$ 的后置节点，记作 $u0\dashrightarrow v1$。

图 4 给出了一个典型示例。由 D 组件可以得到 $d1\rightarrow dd1\rightarrow d2\rightarrow d3\rightarrow dm1\rightarrow d5$ 和 $d1\rightarrow dd1\rightarrow d4\rightarrow dm1\rightarrow d5$。由 A 组件可以得到 $a1\dashrightarrow a2\rightarrow a3\dashrightarrow a4$。

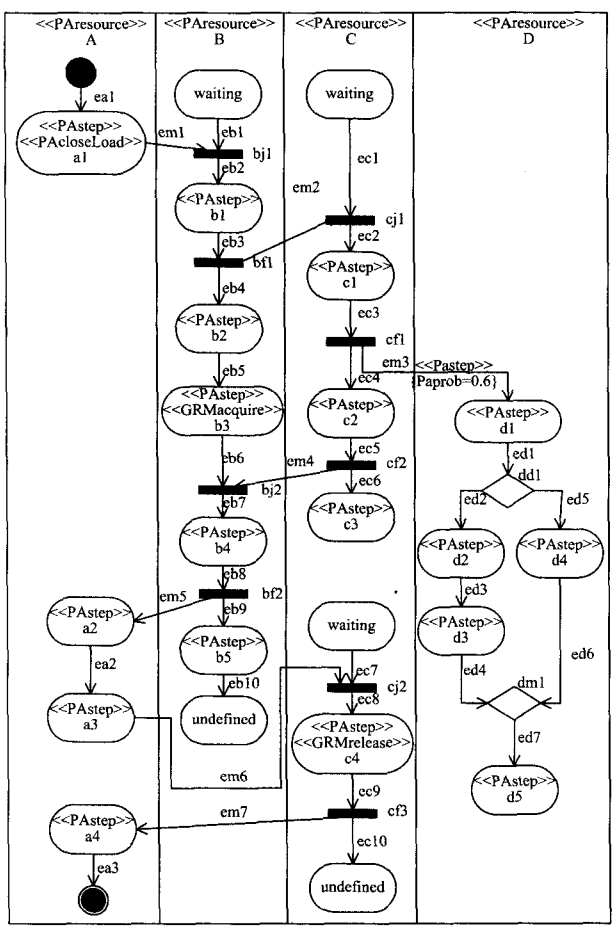

图 4 转换示例

5.2.2 依赖关系分离

组件之间的依赖关系隐含在组件交互的消息序列中，依赖关系分离的目的是将这种关联断开，保持组件内部的连通性。

分离过程是将间接关联节点和异步请求的节点断开，为客户组件添加对应的嵌套活动，并为服务组件添加对应的处理序列（Sequence 及其子类）。服务组件分离后，可能会被资源的释放或转发分割为两个阶段，第一阶段客户组件阻塞等待，第二阶段客户组件不必阻塞。图 5 给出了分离后的示例。

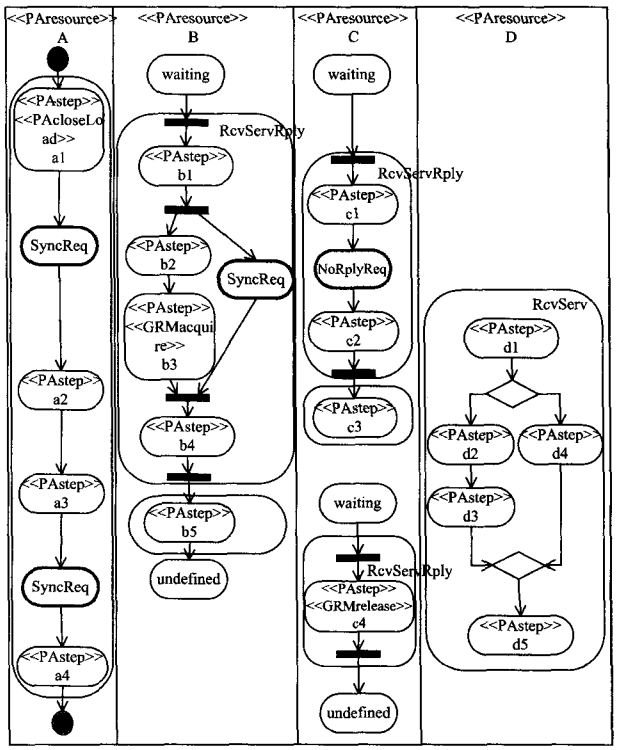

图 5 依赖关系分离

5.2.3 控制流分解

控制流分解主要分析组件内部的控制流，使其由有向图转换为 RSM 模型的树状结构。

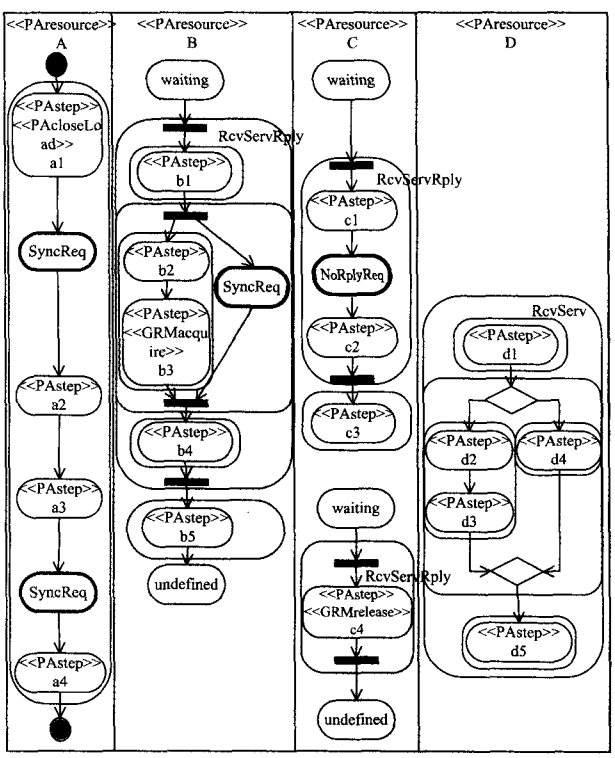

图 6 控制流分解

分析的过程依据节点之间的连通性，按照 Fork/Join 以

及 Decision/Merge 匹配的关系，进行嵌套分析，逐步分解为最小单元。最小单元为只包含一个活动或者一组顺序执行的活动。图 6 给出了分析后的最小单元的示意图。

可以看出 B 组件和 D 组件由于具有分支的存在，因此被分割为多个子区域。这些子区域内部又根据不同的分支分成若干只包含活动节点的最小单元。

A 组件是一个客户组件，其发出的两次同步请求被新增的 SyncReq 节点取代。B 组件则被同步请求分为两个阶段，第一个阶段接受请求，执行操作，并返回消息，此过程中对 C 组件发起的一次同步请求会被 SyncReq 节点取代，第二个阶段做收尾工作。C 组件被先后调用了两次，第一次与 B 组件类似具有两个阶段，而第二次则只有一个阶段。C 组件采用异步调用的方式对 D 组件发出的请求，由 NoRplyReq 节点所取代。D 组件由于接受的是异步调用，不用返回结果，因此只被分为一个阶段。

5.2.4 转换

经过上述转换划分出的单元将按照 RSM 模型进行构造，并对非嵌套依赖和组件的不同连通区域进行特殊处理即可。

对非嵌套依赖，在分析到〈GRMacquire〉标签时，插入一个 Acquire 活动。同样在分析〈GRMrelease〉时插入 Release 活动。由于采用的容器节点描述顺序，因此可以直接插入，不用改变与其它节点的关联关系。

此外，组件可能具有多个功能区域对应多个 Group，如 C 组件所示。特别对于使用 instance 属性保证唯一性的组件，则需要将不同的连通区域合并到 RSM 模型对应的组件中。

图 7 给出了转换后的结果。可以看出，各个组件之间并没有显式的关联，依赖关系被转化为特殊的活动。资源的获取和释放关系，可以很容易地从 RSM 模型中识别。

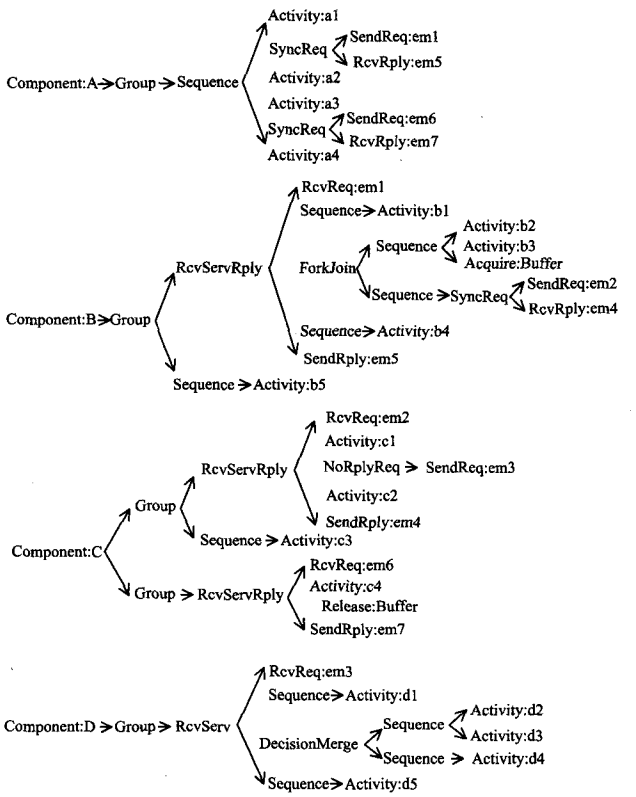

图 7 转换结果

5.3 性能模型转换

性能预测最终需要依靠性能模型进行解析才能得到预测结果。在文献[2]综述中，作者对当前主流的性能模型，以及它们的使用情况作了详尽的分析。在众多模型中排队网模型，受关注程度最高。其中分层排队网模型(LQN)[25]，由于克服了状态空间爆炸，引入了分层概念，以及对资源的良好支持，得到了广泛关注。本文将以 LQN 模型作为主要的性能模型。

分层排队网具有处理器(Processor)和任务(Task)两类资源。任务具有的多个功能接口叫做入口(Entry)。入口可以分为两个或更多的连续阶段(Phase)。入口包含若干活动(Activity)，用于描述执行细节。任务与任务之间存在 3 种不同类型的消息，即同步(Rendezvous)、异步(SendNoReply)和转发(Forward)。

可以看出 LQN 和 RSM 模型有着相似的结构，因此，转换过程比较直接。本文所提出的转换算法分为 3 步：资源转换、嵌套依赖关系转换和非嵌套依赖关系转换。

资源转换将 RSM 所描述的资源转化为 LQN 所对应的资源，包括物理资源和逻辑资源；嵌套依赖转换则负责转换程序执行步骤和嵌套依赖关系；而非嵌套依赖转换则是为性能模型添加非嵌套依赖的关系。表 2 给出了 RSM 模型和 LQN 模型的元模型对应关系。

表 2 RSM 与 LQN 元模型对应关系

类型	RSM 元模型		LQN 元模型
资源	Device		Processor
	Component		Task
基本结构	Group		Entry
	Cycle		Loop
	ForkJoin		And-Fork+And-Join
	DecisionMerge		Or-Fork+Or-Join
	Activity		Activity
依赖关系	SyncReq		Rendezvous
	NoRplyReq		SendNoReply
	RcvServFwd. SendReq		Forward
	Acquire; isblocking	true	Rendezvous
		false	SendNoReply
	Release		Rendezvous

非嵌套依关系的转换相对特殊，因为在 RSM 模型采用了对资源获取与释放的原语，所以在转化为 LQN 模型时需要预构非嵌套模型的模板，以支持原语描述。图 8 给出了该模板样式。

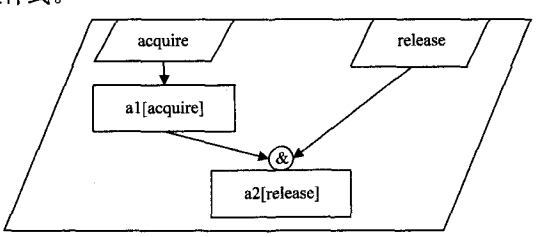

图 8 非嵌套依赖关系模板

RSM 模型与 LQN 模型存在最大的不同在于活动关联关系的描述。但是这两种方式并不冲突，可很容易由容器节点确定活动的链接边的关系。特别地，第一阶段的最后一个消息设为返回活动，用于释放客户组件对其的依赖。

另外，RSM 模型采用特殊的活动和序列描述资源的依赖关系，为了减少创建的活动数，采用了如下的合并规则。

节点 v 是一个请求活动，如果存在节点 u 使得 $u \rightarrow v$，不

存在 w 使得 w 与 v 处于不同分支，且满足 $u \rightarrow w$，那么 v 所对应的请求活动转化为 u 的一个请求边。否则构造一个空活动，其执行时间为 0。

图 7 所描述 RSM 模型转换后的样式如图 9 所示。资源之间的依赖关系转化成了活动与入口的关联边。除了 B 组件请求 C 组件采用了延迟同步，不能合并外，其它请求活动均合并到前驱活动上。UML 模型省略了部署关系，但可从图 9 中看出任务被部署在 3 个不同的处理器上。

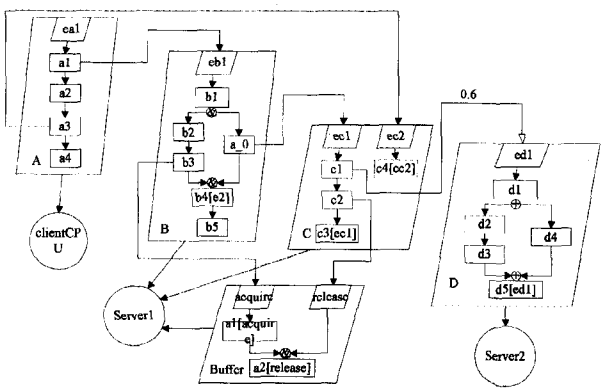

图 9　LQN 模型转换结果

5.5　多模型转换

除了 UML 活动图，软件结构还可以采用 UML 序列图描述，由于序列图与活动图描述的能力相似，转换的方法也类似，本文不详细介绍，图 10 给出了与图 4 相同的模型示例。下面以通用随机 Petri 网(GSPN)[26] 为例，简要介绍 RSM 模型到 GSPN 的转换。

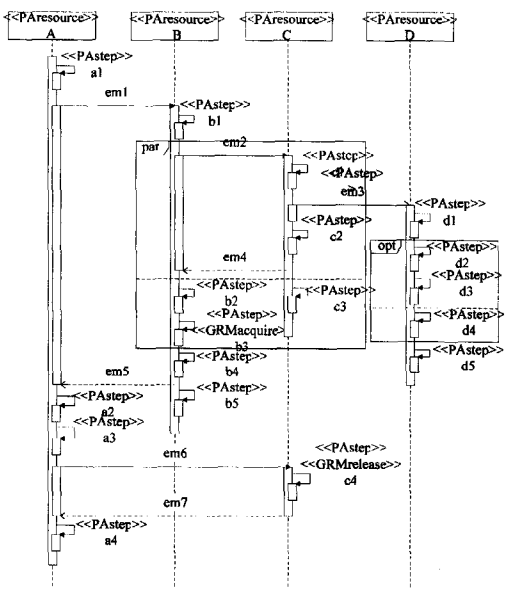

图 10　序列图模型示例

模型的转换可以通过标签 GSPN(LGSPN)网实现[8-10]。LGSPN 模型并没有与 RSM 模型直接对应的关系，但是通过库所和变迁的组合可以建模出与 RSM 模型对等的模型。图 11 给出了预先建立的 LGSPN 模式。特别地，Group 隐含在调用过程中，并未独立给出。

使用模型的转换过程与 RSM 到 LQN 模型的转换类似。模式实例化后，根据转换规则不断地进行组合，则可获得一个完整的 LGSPN 模型。但其与 LQN 模型的转化略有不同，因为 LQN 模型本身具有物理资源的概念，转换不用考虑物理资源的竞争，只需要描述组件的部署情况即可，而 LGSPN 模型不具备这种特性，所以在发生等待让出处理器时，需要释放对物理资源的占用。图 12 给出了图 7 所示 RSM 模型转换后的结果。

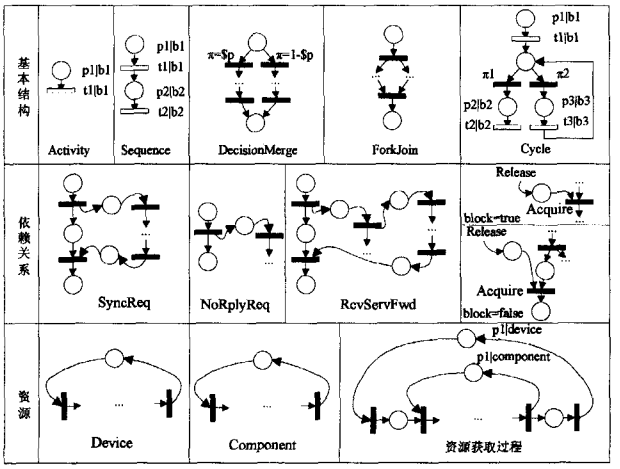

图 11　LGSPN 转换模式

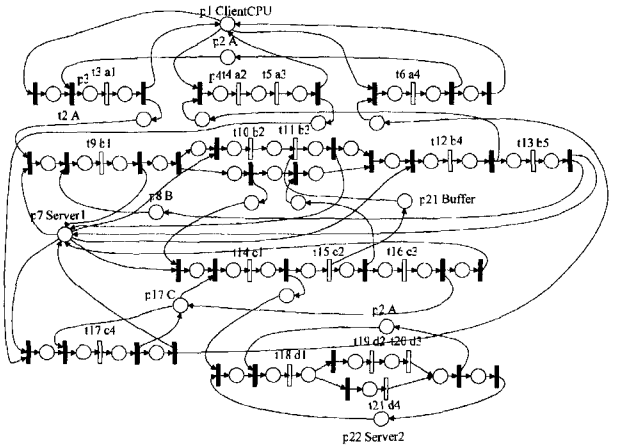

图 12　LGSPN 转换结果

结束语　本文解决的核心是分布式系统的性能建模和模型转换问题，所以并未针对性能模型的求解和结果的分析做出详细说明。本文关注点在于降低建模的复杂度。为此，提出了一种以资源为核心、支持 UML 性能建模与多对多模型转换的方法，以达到降低性能预测方法在软件开发过程中使用的难度，使早期的性能诊断与决策更容易开展。

为了验证本工作，我们设计了名为 OncePD 的原型工具[19]。该工具可支持 UML 模型到性能模型的转换，并能调用相应性能预测工具求解预测结果。该项目的主旨是提供一种简单易行的性能建模方法，以达到性能预测与软件开发过程整合的目的，从而提高软件制品满足性能需求的可信程度。

下一步，我们准备进一步研究资源的自动引入技术，使设计人员可以在不改变传统体系结构设计思想的情况下获得完整的、包含底层资源使用细节的模型。

参 考 文 献

[1] Koziole H. Performance evaluation of component-based software systems: A survey[J]. Performance Evaluation, 2010, 67: 634-658

[2] Balsamo S, Marco D. Model based performance prediction in software development: a survey[J]. IEEE Trans. Softw. Eng.,

2004,30(5),295-310

[3] Object Management Group: Unified modeling language: superstructure, version 2[R]. OMG Adopted Specification, formal/05-07-04 ,2005

[4] Object Management Group: UML profile for schedulability, performance, and time specification[S]. OMG Adopted Specification ptc/05-01-02, July 2005

[5] D'Ambrogio A, Bocciarelli P. A model-driven approach to describe and predict the performance of composite services[C]// Cortellessa V, Uchitel S, Yankelevich D, eds. WOSP. ACM, 2007:78-89

[6] Hillston J, Wang Y. Performance evaluation of UML models via automatically generated simulation models[C]// Jarvis S A, ed. Proceedings of the 19th Annual UKPerformance Engineering Workshop. Warwick, UK, 2003:64-78

[7] Tribastone M, Gilmore S. Automatic Extraction of PEPA Performance Models from UML Activity Diagrams Annotated with the MARTE Profile[C]// Princeton, WOSP. ACM, New Jersey, USA, 2008:67-78

[8] Bernardi S, Donatelli S, Merseguer J. From UML sequence diagrams and statecharts to analysable Petri net models[C]// Proc. 3rd Int. Workshop on Software and Performance (WOSP02). Rome, July 2002:35-45

[9] Lo'pez-Grao J P, Merseguer J, Campos J. From UML Activity Diagrams To Stochastic Petri Nets[C]// Fourth Int. Workshop on Software and Performance (WOSP 2004). Redwood City, CA, Jan. 2004:25-36

[10] Merseguer J. Software performance engineering based on UML and Petri nets[D]. University of Zaragoza, Spain, March 2003

[11] Petriu D C, Shen H. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications [C] // Computer Performance Evaluation / TOOLS. Lecture Notes in Computer Science, Springer, 2002: 159-177

[12] Gu G P, Petriu D C. XSLT transformation from UML models to LQN performance models[C] // WOSP'02. Rome, Italy, July 2002

[13] Koziolek H, Reussner R. A Model Transformation from the Palladio Component Model to Layered Queueing Networks[C]// Kounev S, Gorton I, Sachs K, eds. SIPEW 2008. LNCS 5119, 2008:58-78

[14] Woodside C M, Petriu D C. Performance by unified model analysis(PUMA)[C]// Proceedings of the Fifth International Workshop on Software and Performance, WOSP. ACM, 2005:1-12

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

[16] Mizan A, Franks G. An Automatic Trace Based Performance Evaluation Model Building for Parallel Distributed Systems [C]// Proceedings of the second joint WOSP/SIPEW international conference on performance engineering(ICPE 2011). 2011

[17] Jiang De-jun, Pierre G, Chi C-H. Autonomous Resource Provisioning for Multi-Service Web Applications [C]// 19th proceeding: International World Wide Web Conference. 2010

[18] Zhang Wen-bo, Huang Xiang, Wei Jun. An Aspect-oriented Modeling Approach to Predict Performance of JCA-based Systems[C]// International Conference on Interoperability for Enterprise Software and Applications, I-ESA2009. 2009:140-146

[19] http://oncepd.sourceforge.net/

[20] Woodside M. Software Resource Architecture [J]. Journal of Software Engineering and Knowledge Engineering, 2001, 11(4)

[21] Woodside M. Resource Architecture and Continuous Performance Engineering[C]// Overhage S, et al., eds. QoSA 2007. LNCS 4880, 2007:1-14

[22] Welsh M, Culler D, Brewer E. SEDA: An architecture for well-conditioned, scalable Internet services[C]// Proceedings of the 18th Symposium on Operating Systems Principles(SOSP). October 2001

[23] Cherkasova L, Fu Y, Tang W, et al. Measuring and Characterizing End-to-End Internet Service Performance [J]. Journal ACM/IEEE Transactions on Internet Technology (TOIT), 2003, 3(4):347-391

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

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

[26] Marsan M A, Balbo G, Conte G. Modelling with generalized stochastic Petri nets[J]. ACM SIGMETRISS Performance Evaluation Review, 1998, 26(2)

[27] Krogmann K, Kuperberg M, Reussner R. Using Genetic Search for Reverse Engineering of Parametric Behavior Models for Performance Prediction[J]. IEEE Transaction on Software Engineering, 2010, 36

[28] Woodside M, Li J, Chinneck J, et al. Performance Model Driven QoS Guarantees and Optimization in Clouds [C]// ACM/IEEE ICSE Workshop on Cloud Computing. Vancouver, May 2009

(上接第 146 页)

[11] Li H J, Wang L H, Zhao S, et al. Research on Lossless Compression Algorithms of Low Resolution Palmprint Images[J]. Research Journal of Applied Sciences, Engineering and Technology, 2012, 4(14):2072-2081

[12] Zhang J S, Wang X M, Zhang W F. Chaotic keyed hash function based on feedforward-feedback nonlinear digital filter[J]. Physics Letters A, 2007, 362(5/6):439-448

[13] Weinberger M, Seroussi G, Sapiro G. Lossless image compression algorithm: Principles and standardization into JPEG-LS[J]. IEEE Transactions on Image Processing, 2000, 9(8):1309-1324

[14] http://www4.comp.polyu.edu.hk/~biometrics/

[15] Zhang D, Kong W K, You J, et al. On-line palmprint identification[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003, 25:1041-1050

[16] Mi B, Liao X F, Chen Y. A novel chaotic encryption scheme based on arithmetic coding Chaos[J]. Solitons and Fractals, 2008, 38(5):1523-1531

[17] Li H J, Zhang J S. A secure and efficient entropy coding based on arithmetic coding[J]. Communications in Nonlinear Science and Numerical Simulations, 2009, 14(12):4304-4318

[18] Baptista M S. Cryptography with chaos[J]. Physics Letters A, 1998, 40(1/2):50-54