一种开放环境下的软件可靠性评估方法

陆 文[1),2)] 徐 锋[1),2)] 吕 建[1),2)]

[1)](南京大学计算机软件新技术国家重点实验室 南京 210093)

[2)](南京大学计算机软件研究所 南京 210093)

摘 要 目前,软件系统运行环境日益增强的开放性对原有的软件开发技术(包括软件可靠性评估方法)提出了挑战.一些基于软件测试和模拟的方法由于效率上的不足而不再适用;而另一些方法,例如基于状态的可靠性评估方法,虽然在效率上有了很大的提高,但在适用范围上又存在着不足,比如不能很好地处理含并行结构的系统.为此,在基于状态的可靠性评估方法的基础上,文中提出一种改进的可靠性评估方法,以兼顾效率和适用范围两个方面.首先介绍如何用Petri网来描述各种复杂结构的系统,接着介绍一种自底向上的可靠性计算过程,该过程能对并行结构进行分解和综合计算,高效、准确地计算出系统的可靠性.该方法还可以估算出组件对系统的重要性,从而大大地增强了可靠性评估在软件开发中的作用.

关键词 软件可靠性;开放环境;体系结构;组件重要性

中图法分类号 TP311 **DOI号**: 10.3724/SP.J.1016.2010.00452

An Approach of Software Reliability Evaluation in the Open Environment

LU Wen[1),2)] XU Feng[1),2)] LV Jian[1),2)]

[1)](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)

[2)](*Institute of Computer Software, Nanjing University, Nanjing* 210093)

Abstract Nowadays software environment has been more and more open than before, the openness of environment has greatly influenced many existing software development techniques, including software reliability evaluation. Due to the weakness in efficiency, methods based on testing or simulations are not competent in the open environment; other methods, such as the state-based methods, are more efficient, but are unable to deal with the systems with concurrency. So this paper tries to put forward an improved reliability evaluation method which can do better in both efficiency and applicability. This method first describes how to use Petri nets as software architecture description; and then introduces a bottom-up way to calculate the system's reliability, the calculation process can be done efficiently and accurately, and can also deal with systems with concurrency, which can hardly be handled by existing methods; the method can also generate information reflecting the component's importance to the whole system, which can make software reliability evaluation play a more important role in software development.

Keywords software reliability; open environment; software architecture; component importance

收稿日期:2009-04-06;最终修改稿收到日期:2009-08-04.**陆 文**,男,1984年生,硕士研究生,主要研究兴趣为系统安全、可信计算.**徐 锋**(通信作者),男,1975年生,博士,教授,主要研究兴趣为可信计算、系统安全、软件可靠性技术等.E-mail: xf@nju.edu.cn.**吕 建**,男,1960年生,博士,教授,博士生导师,主要研究领域包括新型程序设计与软件方法学、软件形式化与自动化方法、软件代理、中间件技术等.

1 引　言

在过去的 20 年里，Internet 的迅猛发展大大影响了我们生活的各个方面，这其中也包括了计算机软件的开发和使用. 随着计算机技术，尤其是分布式面向对象技术的发展，如 EJB，.NET 和 Web Services 等[1]，各种软件实体开始以开放、自主的软件服务形式存在于 Internet 的各个节点上，并通过协同机制进行跨网络的互连、互通、协作和联盟[2]，这样，Internet 就逐渐成为一个软件开发、运行和维护的环境和基础.

由于 Internet 本身的开放性和分布性，使得建立在其基础上的软件系统与传统的有所不同，因此会引发一些问题. 其中一个就是：由于对整个环境只有部分的信息并且无法对其进行控制，因而很难确保组合而成的系统具有较高的可靠性，从而很难得到用户的信任. 类似的问题也出现在了 Web Services 的发展与普及过程中，因此，很多工作在这方面展开. Ran 用 QoS 扩展了服务发现来选择具有较高质量的服务[3]；Sahai 等人利用 SLA 的规约来保证服务的质量[4]；Liu 提出了利用信任评估和信任传递来进行服务选择和组装的框架[5]. 然而，这些方法主要涉及的是如何对构成系统的组件进行评估，而对于组合而成的系统的可靠性则鲜有涉及. 而在传统的方法中，基于测试和模拟的方法由于要进行详尽的分析和计算，因而其效率比较低下，很难适应动态多变的开放环境；基于体系结构的可靠性评估方法[6-9]虽然在效率上有所进步，然而它们在 Internet 这种环境下还存在着缺陷，一方面这些方法很难处理带并行结构的系统[10]，其所采用的软件体系结构描述方法也不能很好地描述这种结构，因而在适用范围上还存在着缺陷；另一方面它们的效率还需要进一步提高，尤其是在增量更新方面；同时，这些方法主要侧重于对系统的评估结果，很少能产生关于组件的评估信息，因而在软件开发过程中只能提供有限的信息.

为了解决上述问题，本文试图对一种特殊的基于体系结构的可靠性评估方法（即基于状态的方法）进行改进. 一方面，利用开放环境下的软件系统结构简单明了的特征，通过一种自底向上的方式来计算可靠性，以进一步增强效率和增量计算的能力；另一方面，通过对系统中的并行结构进行分解、并分别加以计算、最后再综合的方式来增强方法的适用范围；同时，在整个计算过程中，还利用计算出来的各部分的可靠性信息来计算每个组件的重要性，来增强方法的作用. 通过这 3 个方面的改进，可以使得开发并运行在 Internet 上的软件系统的可靠性可以被准确快速地计算出来，并能随着环境和软件系统本身的变化而变化，同时组件重要性这一参数还能为系统的演化过程提供有效的参考.

本文第 2 节介绍相关工作，并阐述传统方法的不足以及对改进方法的要求；第 3 节给出一个利用 PetriNet 来描述软件体系结构的方法；第 4 节介绍详细的可靠性评估过程；第 5 节给出一个简单的例子来进一步展现这个评估过程，并验证方法的有效性和正确性；最后，对本文进行总结并对后期工作进行展望.

2 开放环境下现有软件可靠性评估技术的分析

过去的 30 年里，在软件可靠性评估这一领域已经有了很多的方法和技术. 其中一种是黑盒的方式，这种方式将整个系统看作一个单一的个体并且不考虑其内部结构，例如各种软件可靠性增长模型[11-12]，这些模型通过软件测试中得到的一些统计数据来预测软件的可靠性，由于对软件测试的依赖，使得其在效率上和可操作性方面都不适合开放环境的需要；另一种是白盒的方式，这种方式会根据软件系统的体系结构信息来将各个组件的可靠性进行综合计算，以得到系统的可靠性[6-9].

白盒的方式又可以进一步地分为两类：基于路径的方法和基于状态的方法[6]. 基于路径的方法通过计算软件所有可能执行路径的可靠性，然后加以综合来得到整个软件的可靠性，但这种方法不适合于具有无限路径的系统，因而存在着一定的局限性，尽管 Dolbec 和 Shepard 试图采取用组件的使用率来计算可靠性这一方法来解决这个问题，但会带来精度上的损失[9]. 相对来说，基于状态的方法则更优越并且更加流行一些，它们利用控制流程图来表示软件系统的内部结构，并用分析的方式来计算整个系统的可靠性[7]. 这些方法通常有两个假设：组件间的独立性和组件间控制转移的 Markov 性，前者表示组件间的失效行为是相互独立的，而后者则表示控制的转移只取决于当前的状态，而不受历史行为的影响. 这两个假设在大部分情况中都很难满足，因而在适用范围上有着局限性. 幸运的是，这些假设在开放环境下是常常得到保证的，因为开放环境下的组件是松耦合的，因而具有较高的独立性. 因此，和

其它的方法相比，基于状态的方法更加适合用来解决开放环境下软件可靠性的评估问题.

然而，基于状态的方法实际上还存在着一些不足.由于这些方法最初是在过程化的程序设计时代产生的，因而常常假设在任一给定的时刻，只有一个组件正在执行，因此这些方法无法适用于多个组件同时运行的情况[10]，而这些情况在开放环境下是经常出现且不可避免的；更糟糕的是，它们常常使用控制流程图来表示程序的结构，这就使得对并行结构的描述很不方便，就更不用说在此基础上进行进一步的分析了.因此从适用范围来说，基于状态的方法无法完美地适应开放环境.

除此之外，开放环境还在其它方面带来了一些新的需求：

(1)较高的效率.在开放环境中，软件系统是由不同的组件动态并临时的组合而成，因此对其进行详细彻底的分析是不实际、不经济的；同时，环境的多变性也要求评估方法能尽快完成，以保证所得结果依然是有效的.因此，效率是最重要的.

(2)足够的精确度.高效率的要求必然对评估结果的精确度带来了影响，但评估方法仍然要给出足够精确的结果，不然会使得用户得到错误的信息，做出错误的判断.

(3)能产生关于组件的评估结果.开放环境下的系统的典型特征是，整个系统的功能是要靠各个独立组件之间的协作和联盟来实现的，因而组件所占据的地位越来越重要.因此，开放环境下的可靠性评估方法还应该提供关于组件的信息来帮助系统的优化和改进，从而在多变的环境中保持较高的质量.

然而现有的方法通常通过一个详细的分析来得到非常精确的结果，因此在效率上比较低下，这与开放环境下更加看重效率的要求正好相反.同时，它们往往只得到关于系统可靠性的信息，而很少产生关于组件的信息.

本文将试图解决这些问题.

3 软件体系结构的描述

软件的体系结构包含了对构成软件系统的组件的描述，组件间的交互、组件间组合的模式以及对这些组合模式的约束[13]，这些都为针对整个软件系统的可靠性评估提供了有益和必需的信息.一方面，它包含了整个系统的结构信息，而系统的结构是软件可靠性的一个重要影响因素[14-15]；另一方面，它还展示了系统在运行时刻是如何活动的动态信息，这是我们用来预测软件运行时刻行为(包括软件可靠性)的重要依据.

要想充分并有效地利用体系结构信息，就需要使用一种合理并方便的描述手段.一般来说，体系结构的描述能力会影响到基于其的评估方法的适用范围和易用程度.例如，Wang 等人用状态图来表示体系结构[8]，它们的方法只适用于一些预定义的经典的结构风格；Gokhale 提出的可靠性评估方法无法处理带并行结构的系统，其使用的带概率的控制流程图[7]也很难对并行这种结构进行描述；OWL-S 使用了几种基本的控制结构来描述组合服务的过程模型，这种方法可以描述绝大部分系统的结构[16]，但对一些复杂结构(例如递归、嵌套循环的跳出)却束手无策.考虑到传统方法的不足，本文借鉴了 Aalst 在工作流方面的工作[17]，决定尝试用 Petri 网来作为体系结构的描述工具.Petri 网看起来与流程控制图很像，但它能处理包括并行结构在内的各种复杂结构，并且是一种被广为研究和使用的工具.

图 1 是一个用 Petri 网来描述软件体系结构的

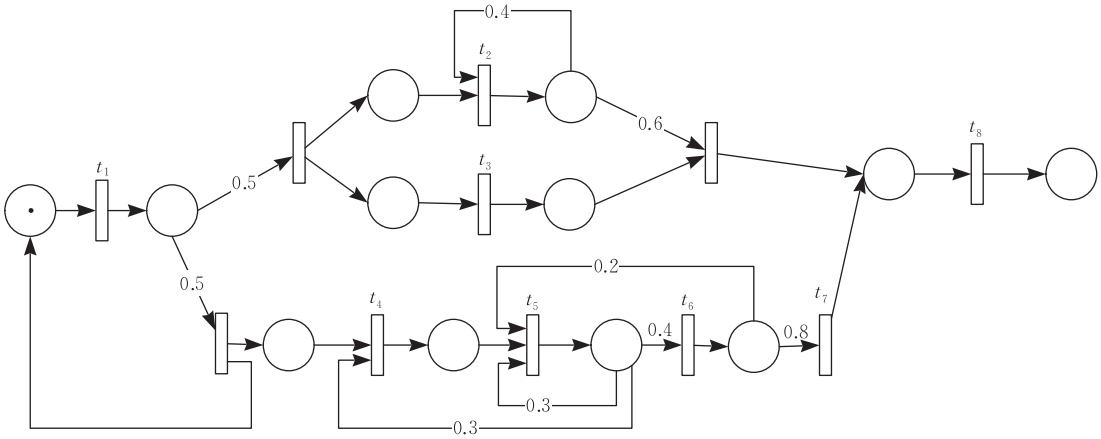

图 1 体系结构描述示例

例子.其中 Petri 网中的绝大部分变迁用来描述软件的各个组件(还有一部分变迁用作控制结构,如后文所述),在这里,变迁也可以代表一个子系统,当需要查看系统的简化模型时,可以将代表子系统的图形部分化归为一个变迁;反之,当要查看某个子系统的细节内容时,可以用更细节的内容代替相应的变迁.在体系结构的 Petri 网描述中,库所表示了不同组件之间控制转移的中间状态,在某个库所里有一个令牌表示控制将转移到后续的组件,显然,令牌在图中的传递过程代表了运行时刻软件的执行路径.

在实际使用中,体系结构的 Petri 网表示形式可以利用几种典型的结构通过自底向上的方式构造而成:

(1)顺序

顺序是最常用的一种结构,它代表了各个组件依次执行,图 2(a)是顺序结构的表示方式.

(2)分支

分支表明程序将从几条候选路径中选出一条来执行,而具体选择哪一条,取决于运行时刻的程序上下文信息.显然这种软件系统运行时刻的行为会对整个系统运行的成功与否有着明显的影响,但是我们却无法预测到这种行为,不过我们可以估计每个分支被执行的概率,这些概率足以让我们对系统的可靠性进行评估.图 2(b)是分支的描述形式,每个分支的概率都标注在相应的弧上.

(3)循环

循环常常被用来表示某个组件将被重复的执行.图 2(c)描述了循环结构.在这里,也有一些概率信息,这些信息表示控制流离开和回到循环体的概率.

(4)并行

并行表示几个组件都会被执行.在图 2(d)中,变迁 B 和 C 表示两个会被一起执行的组件;而变迁 A 和 D 则用来描述程序执行路径的分裂和合并,其中 D 不是必须的,这表示所有的路径一直延续到结束,这种情况也是比较常见的,例如 Unix 类系统中的 fork 调用会造成这种结果.

(5)调用

调用也是一种经典的结构,它出现在各种程序开发方法中,如过程式编程中的函数,面向对象中的方法等,并且是分布式环境中组件间相互交互的一种主要手段.而对调用的描述也比较复杂,假设组件 A 调用了组件 B,如果知道 A 的内部结构,那么我们可以用这个内部结构的 Petri 图表示替代对应 A 的变迁;否则,我们可以简单地把 A 和 B 看成是顺序执行的,这将不会影响到后面对可靠性的评估结果.

(6)goto

虽然 goto 是一种不推荐的控制结构,但其仍有一定的价值.因为有一些复杂结构,例如递归、异常处理等,很难用上面的 5 种结构来表示,而用 goto 来表示就比较简单和方便.goto 的 Petri 网表示也比较简单,只是将跳转的组件之间用一个库所连接起来即可,因此可由图 2(a)来表示.

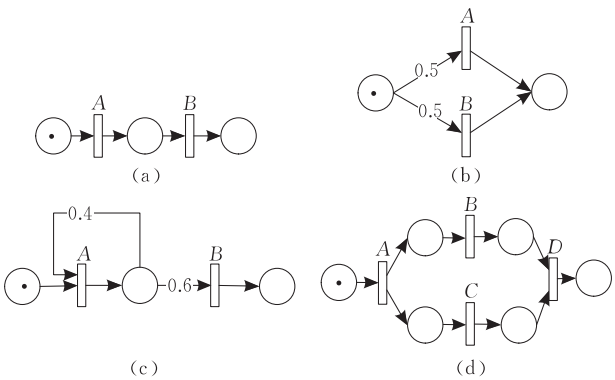

图 2　几种典型结构的 Petri 网描述

通过这几种控制结构,我们便可以绘制出描述整个软件系统的 Petri 网来,通常,我们在最前端和最后端各增加一个库所,分别来表示软件系统的开始和结束状态.图 1 便描述了由 8 个组件组成的软件系统.

从图 1 可以发现,Petri 网描述在图形上与控制流程图很相似,因此一些基于流程图的方法可以很容易地转移到 Petri 网上面来;而在 Petri 网方面,已经有很多工作用于系统行为属性的分析与验证[17-18],这些工作可以作为系统评估的一部分来提供更多关于系统的信息,因而可以使得我们的方法和这些工作能很好地共存,并作为对软件系统进行全面分析的一部分.同时,这种图形化表示方法也能很容易地从实际的系统中得到,以及与其它表示方法相互转换,如 OWL-S[18].

而在体系结构的提取方面,由于开放环境下的软件系统中,组件具有较高的独立性,它们之间的关系也呈现出松散耦合的特性,因而我们能很容易地从软件系统的实现或者规约说明中分辨出不同的组件,并分析出它们之间的关系,从而得到类似图 1 那样的 Petri 表示,但其中的分支概率这类体现软件运行时刻行为的动态信息,则很难直接获取.在传统的软件开发中,这些概率信息可由软件测试来得到,而在开放环境的软件系统中,测试往往是不可行的.这时候,我们可以在系统运行的初期设置一个初始

值,这些初始值可以由专家来指定,或者是一个预定的值.在程序的运行阶段,我们可以监控并统计系统的运行行为.当积累的信息达到一定程度时,可以利用这些充足的、准确的信息对系统进行重新评估以获得更准确、更实时的结果,因此可以说,本文提出的可靠性评估方法在软件系统的运行时刻会产生出更加准确、有意义的结果.

4 软件可靠性评估方法

在本节,我们将首先介绍可靠性评估方法所生成的各种结果,然后再详细介绍如何来得到这些评估结果.

4.1 评估结果

4.1.1 对系统的评估结果

首先,评估方法必须给出关于系统的可靠性.在我们的方法中,采用了悲观估计的方式来估计软件的可靠性.

定义 1. 软件系统的可靠性是对整个系统成功执行的期望,而系统的成功执行意味着所执行的所有组件必须全部按用户的期望执行(需要注意的是,冗余备份中的组件可以允许部分失效,这时可把整个冗余备份结构看成一个组件).假设系统由 Ω 来表示,组件为 $C_1,C_2,\cdots,C_n$,各个组件在系统一次运行过程中的运行次数为变量 $t_1,t_2,\cdots,t_n$,则系统的可靠性 $R(\Omega)$ 可由式(1)给出:

$$R(\Omega)=E\left(\prod_{i=1}^{n}R(C_i)^{t_i}\right) \tag{1}$$

由于开放环境中各个组件之间往往是相互独立的,因此可以对公式进行简化来得到式(2),以使得在整个计算过程中可以对各个组件分别进行计算,这样就大大降低了计算的复杂程度.

$$R(\Omega)=\prod_{i=1}^{n}E(R(C_i)^{t_i}) \tag{2}$$

由该式(2)可以看出,系统的可靠性取决于各个组件的可靠性和它们的平均运行次数,而平均次数的获取是比较麻烦的,下面将介绍如何将这部分工作进行简化.

4.1.2 对组件的评估结果

开放环境下的可靠性评估还需要考虑各个组件对系统可靠性的影响,即组件的重要性.文献[19]对此进行了简单的分析,可以看出组件的重要性主要取决于组件自身的可靠性和运行剖面(主要是组件的运行次数),但该文并没有给出系统的计算方法.文献[20]通过方差来计算组件的重要性,但系统的成功失败是一个伯努利随机变量,这一类变量的方差是由期望决定的,因此方差这种指标只侧重于组件可靠性大小对系统可靠性的影响.Siegrist 提出了用偏导数来表示组件的重要性[21],但其采用的计算方法仅仅考虑了每个组件的平均运行次数,而不考虑它自身的可靠性大小,因而无法给出全面的结果.

在本文提出的方法中,依然采用了偏导数来描述组件的重要性,其定义如下.

定义 2. 软件系统中组件的重要性是指系统可靠性相对于该组件可靠性的变化率,其值越大,则说明该组件对系统的影响越大,因而具有较高的重要程度.假设系统由 Ω 来表示,则组件 A 相对于该系统的重要性 $D(\Omega,A)$ 可由式(3)得出:

$$D(\Omega,A)=\frac{\partial R(\Omega)}{\partial R(A)} \tag{3}$$

从该式(3)可以看出,所产生的重要性首先是依赖于组件的运行次数的,因为软件的可靠性是由组件可靠性作为变量的函数,其具体的计算是依赖于其运行次数的,因而最后得到的偏导值也受其影响;另一方面,组件可靠性的改变虽然不会使得自身重要性产生变化,但会影响到其它组件的重要性计算,因而在组件之间的比较中仍会产生影响,因此可以说组件可靠性大小会间接影响其在整个系统中的重要程度.在下一节,我们将通过对实验结果的分析来验证这一点.

4.2 评估过程

4.2.1 系统的分解

众所周知,现有的软件系统是以一种自底向上的方式逐层地构造而成,而每一层的构造方式都比较简单,呈现出几种典型的模式.直观上来看,可以按照软件系统的构造方式来逐层的计算系统的可靠性,这样能大大减少了计算规模和计算复杂度.

假设系统 Ω 包含组件 $C_1,C_2,\cdots,C_n$,各个组件的运行次数变量为 $t_1,t_2,\cdots,t_n$,而这 n 个组件又可分别组成 m 个子系统 $S_1,S_2,\cdots,S_m$,这 m 个子系统的运行次数变量为 $u_1,u_2,\cdots,u_m$,则根据式(1),我们可以得到子系统 S_i 的可靠性为

$$R(S_i)=E\left(\prod_{C_k\in S_i}R(C_k)^{\frac{t_k}{u_i}}\right),$$

同时,整个系统的可靠性可用如下方式进行计算:

$$\begin{aligned}R(\Omega)&=E\left(\prod_{i=1}^{n}R(C_i)^{t_i}\right)\\&=E\left(\prod_{j=1}^{m}\left(\prod_{C_k\in S_j}R(C_k)^{t_k}\right)\right)\end{aligned}$$

$$=E\Big(\prod_{j=1}^{m}\Big(\prod_{C_k\in S_j}R(C_k)^{\frac{t_k}{u_j}}\Big)^{u_j}\Big)$$

$$\approx E\Big(\prod_{j=1}^{m}\Big(E\Big(\prod_{C_k\in S_j}R(C_k)^{\frac{t_k}{u_j}}\Big)\Big)^{u_j}\Big)\qquad(*)$$

$$=E\Big(\prod_{j=1}^{m}(R(S_j))^{u_j}\Big)\qquad(4)$$

其中(*)的推导可参考文献[20]的附录. 同时，式(3)也可写为

$$D(\Omega,C_i)=\frac{\partial R(\Omega)}{\partial R(S_j)}\cdot\frac{\partial R(S_j)}{\partial R(C_i)},\ C_i\in S_j\qquad(5)$$

4.2.2　简单子系统的计算

在软件的构造过程中，最常见的是几种典型的构造方式，可以对这几种构造方式分别进行讨论以得到对式(2)和式(3)进行特化后的结果.

(1) 顺序结构

在顺序结构的子系统，各个组件依次执行，因此，每个组件都执行一次. 假设子系统 S 由组件 C_1，C_2，…，C_n 构成，则运用公式(2)和(3)可得到

$$\begin{cases}R(S)=\prod_{i=1}^{n}R(C_i)\\ D(S,C_i)=\sum_{j=1}^{n}\Big[D(C_j,C_i)\cdot\prod_{k=1,k\neq j}^{n}R(C_k)\Big]\end{cases}\qquad(6)$$

(2) 分支结构

在分支结构中，每次运行时，会以一定的概率选择其中一个分支来运行，假设子系统 S 由组件 C_1，C_2，…，C_n 构成，各组件执行的概率为 $P(C_1)$，$P(C_2)$，…，$P(C_n)$，而这些概率信息又代表了每个组件平均被运行的次数，因而可得

$$\begin{cases}R(S)=\sum_{i=1}^{n}[R(C_i)\cdot P(C_i)],\\ D(S,C_i)=\sum_{j=1}^{n}[D(C_j,C_i)\cdot P(C_j)],\\ \text{其中}\ \sum_{i=1}^{n}P(C_i)=1\end{cases}\qquad(7)$$

(3) 循环

循环可分为两种：一种是固定次数的循环，表示循环体执行的次数是确定的，假设循环体 A 会被执行 n 次，则可以得到

$$\begin{cases}R(S)=R^n(A)\\ D(S,A)=n\cdot R^{n-1}(A)\end{cases}\qquad(8)$$

还有一种循环是基于条件控制的循环，循环体会一直被执行直到某个终止条件被满足. 假设循环体为 A，控制流返回 A 的概率为 $P(A)$，则我们得到整个子系统 S 的可靠性以及 A 的重要性为

$$\begin{cases}R(S)=\dfrac{1}{1-R(A)\cdot P(A)}\cdot R(A)\cdot[1-P(A)]\\ D(S,A)=\dfrac{1-P(A)}{[1-P(A)\cdot R(A)]^2}\end{cases}\qquad(9)$$

(4) 并行

并行结构的可靠性很难判断，因为这个结构本身并没有明确的定义，有的情况下并行结构要求所有的组件都要成功完成，而有的情况下只要求其中的几个组件成功完成. 鉴于这个原因，本文仅对其中两种基本的种类进行讨论.

第 1 种是与并行，其含义是只有当并行结构的所有组件都成功运行时，整个并行结构才能称为是成功的，这经常出现在为提高性能而采取并发操作的情况下. 假设系统 S 由组件 C_1，C_2，…，C_n 构成，则可用式(6)计算出与并行的可靠性和其中组件的重要性.

第 2 种是或并行，它要求只要有一个组件能成功运行即可. 假设子系统 S 由组件 C_1，C_2，…，C_n 构成，则可由式(10)计算出或并行的可靠性和其中组件的重要性：

$$\begin{cases}R(S)=1-\prod_{i=1}^{n}[1-R(C_i)]\\ D(S,C_i)=\sum_{j=1}^{n}D(C_j,C_i)\cdot\prod_{k=1,k\neq j}^{n}[1-R(C_k)]\end{cases}\qquad(10)$$

与并行和或并行是两个基本的并行结构，其它的复杂的情况，例如“至少两个组件成功”等，都可以通过与并行和或并行的复合来得到.

4.2.3　复杂子系统的计算

在大多数的情况下，软件系统每层的构造方式可以由上述几种模式来描述. 然而还存在其它一些复杂的结构难以应付，主要是一些“goto”类型的控制结构，例如嵌套循环的跳出、异常处理等. 在这种情况下，我们可以利用传统的基于状态的可靠性评估方法来处理，当然还需进行一些改进来应对传统方法无法处理带并行结构系统的问题.

总的说来，这部分计算包括了 3 个部分：

(1) 将系统分解为一系列的“进程”.

为了使用基于状态的方法，首先应该把系统里的并行部分分离出来，单独计算. 为了达到这个目的，我们可以把每个最外层的并行结构(意指不包含在其它并行结构里)用一个虚组件来替代，这样我们就得到了一个不含并行结构的系统，称之为“进程”.

而对每个虚组件来说,其并行结构的每个部分都是相互独立的,可以递归的转化为一系列的进程.详细的过程如下:

1. 对系统的 Petri 网图进行处理,首先找到最外层的并行结构,这可以通过首先找到从起始库所出发所遇到的第一个分裂变迁(可能有多个,可对每个分别处理)和其对应的合并变迁(如果没有,则以结束库所代替),例如图 2(d)中的 A 和 D,然后找到所有源自于分裂变迁但又出现在合并变迁前面的各个库所、变迁和弧,这些便是最外层并行结构所包含的部分,例如在图 1 中,所有在变迁 t_4 和变迁 t_7 之间的部分.用一个变迁来代替这个并行结构,用来表示虚组件.这样,我们就得到了一个不含并行结构的 Petri 网图,可称为进程,可以对这个进程中的所有成分加以标记,以表明这些部分已经出现在一个生成的进程中了.然后进入步 2.

2. 虚组件的内部结构可以被分为几条独立的路径,对每条路径,我们可以识别出与其相关的库所、变迁和弧,这些内容便是该路径的 Petri 网表示,通过步 3 来对每条路径进行处理.

3. 此步所做的工作与步 1 相同,只是稍微有点差别,因为在处理过程中可能会碰到一些已经出现在其它进程里的变迁或库所.假设我们在试图构造进程 P_i,在构造过程中遇见了一个库所 p(也可能是一个变迁,在这种情况下,我们可以在这个变迁前面加一个辅助的库所 p),而 p 已经出现在了先前生成的进程 P_j 中.这时,我们对 P_j 的内容进行分析,找出所有源自于且仅源自于 p 的变迁、库所和弧,这一部分可以被单独地作为一个进程,记作 P_k,同时 p 出现在 P_i 和 P_j 的部分也可以用 P_k 来代替.

通过以上的工作,我们可以从原始的系统表示中得到一系列的进程和虚组件,这些进程和虚组件也有相应的 Petri 网表示.图 3 表示了对图 1 进行分析后的结果,可以看出所做的工作仅仅是对原图进行了划分,分成几个相对独立的部分(当然还要添加一些额外的库所和变迁来保证每部分是正确的 Petri 图).在图 3 中,生成的进程在左边,而虚组件的表示在右边.

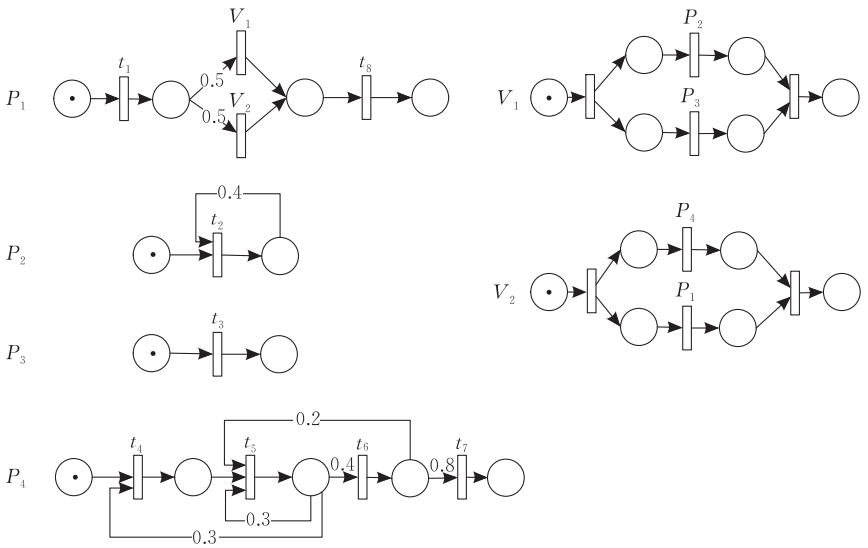

图 3 生成的虚组件和进程

(2)计算每一个进程的可靠性.

这一步将利用基于状态的方法来计算每个进程的可靠性.假设进程 S 由组件 $C_1, C_2, \cdots, C_n$ 构成,可以得到组件之间的转移概率矩阵 $\boldsymbol{Q}$,其中 Q_{ij} 表示组件 C_j 在组件 C_i 之后紧接着执行的概率.可以看出 $\boldsymbol{Q}^k$ 为执行 k 步之后的转移概率.可以证明当 t 趋于无穷大时 $\sum_{k=0}^{t}\boldsymbol{Q}^k$ 收敛,因此就存在矩阵

$$\boldsymbol{M}=(\boldsymbol{I}-\boldsymbol{Q})^{-1}=\boldsymbol{I}+\boldsymbol{Q}+\boldsymbol{Q}^2+\cdots=\sum_{k=0}^{\infty}\boldsymbol{Q}^k.$$

假设组件 C_1 是代表起始状态的组件,而 C_n 是代表终止状态的组件,则元素 M_{1i} 表示组件 C_i 的平均执行次数,利用这个信息,我们可以计算进程的可靠性:

$$\begin{aligned}R(process)&=E\Big(\prod_{i=1}^{n}R(C_i)^{t_i}\Big)\\&\approx\prod_{i=1}^{n}R(C_i)^{E(t_i)} \qquad (**)\\&=\prod_{i=1}^{n}R^{M_{1i}}(C_i)\end{aligned}$$

$$D(process,C_i)=\sum_{j=1}^{n}\Big[D(C_j,C_i)\cdot M_{1j}\cdot R^{M_{1j}-1}(C_j)\cdot\prod_{k=1,k\neq j}^{n}R^{M_{1k}}(C_k)\Big] \qquad (11)$$

这里(**)的推导可参考文献[20].

(3)利用进程的可靠性来计算整个系统的可靠性.

在大多数情况下，进程之间的关系比较简单，可以用一棵树来表示，在这种情况下，系统的可靠性可以利用前面的方法，按照树的结构自底向上的计算来得到. 然而在极少数情况下，进程之间的关系比较复杂，很难用一棵树来表示，例如，在图 3 中，可以看出对 P_1 可靠性的计算依赖于 V_2 的可靠性，而 V_2 可靠性的计算依赖于 P_1 的可靠性，这种互相包含以及可能出现的自包含的关系将使得逐层的计算方法束手无策.

在这种情况下，假设我们用 V_i 来代表虚组件，用 P_i 来表示进程，C_i 表示各个原子组件，则进程的可靠性计算可以由式(11)来表示，而虚组件的可靠性可由式(6)和式(10)来表示，这样，我们就得到了一个方程组：

$$\begin{cases} R(P_1)=f_1[R(V_1),\cdots,R(V_n),R(C_1),\cdots,R(C_n),\\ \qquad R(P_1),\cdots,R(P_n)] \\ R(P_2)=f_2[R(V_1),\cdots,R(V_n),R(C_1),\cdots,R(C_n),\\ \qquad R(P_1),\cdots,R(P_n)] \\ R(P_n)=f_n[R(V_1),\cdots,R(V_n),R(C_1),\cdots,R(C_n),\\ \qquad R(P_1),\cdots,R(P_n)] \\ R(V_1)=g_1[R(P_1),\cdots,R(P_n)] \\ R(V_2)=g_2[R(P_1),\cdots,R(P_n)] \\ R(V_n)=g_n[R(P_1),\cdots,R(P_n)] \end{cases}$$

通过解这个方程组，就可以得到各个虚组件以及进程的信任值了. 而对重要性的计算也可以采用这种列方程组求解的方式.

4.2.4　综合计算

前面描述了对各种控制结构进行计算的方法. 这样，当我们把整个软件的结构用一棵树来表示(其中根节点代表整个系统，中间节点表示具有以上各种控制结构的子系统，而叶节点是各个原子组件)，就可以按照公式很容易地自底向上计算出整个软件系统的可靠性 $R(\Omega)$. 而每个组件的重要性也能很容易地计算出来，假设 a 是一个原子组件，整个系统 Ω 在最上层是由组件 $C_1,C_2,\cdots,C_n$ 构成的，那么可以利用式(12)来计算 a 在整个系统中的重要性 $D(\Omega,a)$，这里 $D(C_i,a)$ 可以被递归地计算出来.

$$D(\Omega,a)=\sum_{i=1}^{n}[D(\Omega,C_i)\cdot D(C_i,a)] \quad (12)$$

4.2.5　可靠性计算方法小结

从上面的分析可以看出，通过对并行结构以先分解再综合的方式进行处理，使得传统的基于状态的评估方法在适用范围上得到了增强.

而效率方面，对简单子系统的处理中，由于有效地利用了开放环境下系统结构简单清晰的特点，计算复杂度是 $O(n)$(n 是组件的个数)；而对复杂子系统处理中，采用的是基于状态的评估方法，因而计算复杂度依然是 $O(n^3)$，这是由于主要的资源会被消耗在逆矩阵的计算上. 由于在整个系统中，复杂子系统出现的概率比较低，因此可以近似地认为整个方法的复杂度是线性的，这样，就比单纯地采用基于状态的方法要高效得多.

这个方法同时也非常适合于多变的开放环境所要求的增量更新. 当某个组件的可靠性发生了变化，这时只需要对代表那个组件的叶节点到根的路径进行重新计算，因而计算复杂度是 $O(\log n)$，而在传统的方法中，则往往需要从头开始计算；同时，我们可以忽视那些重要性较低的节点的变化，因为它们的改变不会对系统造成实质性的影响，这一点在我们的实验结果中可以看出，通过这种集中注意力在少数组件上的方式，可以进一步增强增量更新的效率. 而当系统的结构发生了改变，如果这个变动较小，那么只需要对包含这部分结构的子树进行重新估算即可，这样只需要做少量的工作. 而最差情况下，系统的结构变动非常大，这时就必须要对整个系统进行重新评估.

5　实验与分析

本节将针对图 1 所表示的系统给出可靠性评估的示例，计算出系统的可靠性和各个组件的重要性，并对结果加以分析，以考察本文所提出方法的正确性和有效性. 在这个例子中，位于上面部分的并行定义为一个与并行，而下面一个定义为或并行，这样这个例子就包括了各种控制结构.

可以看出，该系统最顶层的结构是非常复杂的，所以我们对其采用复杂子系统的计算方法. 在进行了第 1 步的工作后，我们可以得到如图 3 所示的一些进程和虚组件，其中 P_1，P_2，P_3 和 V_1 的结构都比较清晰，可以用简单子系统的计算方法来处理；而 $P4$ 的结构并不清晰，但其没有并行部分，因此可以直接采用复杂子系统计算方法中的第 2 部分来进行计算.

假设每个组件的可靠性为 0.9，那么我们可以构造出代表 P_1(即整个系统)的一棵树来，如图 4 所示. 通过自底向上的计算，我们得到树中各个内节点的可靠性表示，并标在每个节点的旁边. 最后，我们

可以得到 P_1 的可靠性，即

$$R(P_1)=0.405\cdot(0.7594+R(V_2)) \quad (13)$$

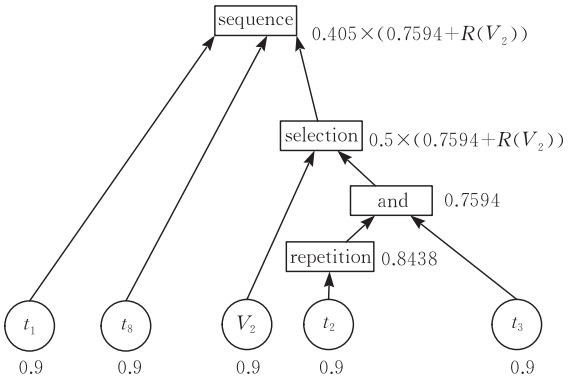

图 4 系统的树状结构

接下来对 P_4 进行处理，通过计算我们可以得到 t_4，t_5，t_6 和 t_7 的平均执行次数分别为 1.9375，3.125，1.25，1. 这样，P_4 的可靠性可以用式(11)来计算，结果为 0.4628. 同时，V_2 的可靠性可以用式(10)计算，结果为

$$R(V_2)=1-(1-0.4628)\times(1-R(P_1))$$
$$=0.4628+0.5372\times R(P_1) \quad (14)$$

式(13)和式(14)构成了一个方程组，通过简单的计算，我们可以得到最终系统的可靠性为 0.6326，这与模拟实验中得到的结果 0.6393 非常接近，这说明整个方法在提高了效率的同时，只会引起很小幅度的精确度下降.

而组件重要性也可以用类似的方法计算出来，表 1 列出了计算得出的每个组件的重要性. 可以发现，这些结果与我们的直观感觉保持一致，t_2 比 t_3 执行得更加频繁，因而具有较高的重要性；t_6 是在一个 or 并行结构里面，而这个结构通常是用作冗余备份的，从而可以减轻其组件失效的影响，可以从表 1 中看出，t_6 的重要性要相对于其它组件低得多；在表格的第 3 行描述了当某个组件的可靠性改善 5% 之后整个系统可靠性的提升幅度，可以看出对重要性高的组件的改进会给系统可靠性带来更大的提高，同时那些重要性比较低的一些组件(如 t_6 和 t_7)对系统的影响很微小，因而可以在更新计算中省略，这能减轻系统监控的负担，并节省计算资源.

表 1 组件重要性及其改进后对系统的影响

组件	重要性	可靠性的改进/%
t_1	0.8984	8
t_2	0.6824	7.9
t_3	0.4368	4.8
t_4	0.1895	2.8
t_5	0.3056	3.3
t_6	0.1223	1.9
t_7	0.0978	1.7
t_8	0.8984	8

图 5 和图 6 进一步分析了重要性这一参数. 图 5 表示了当 t_2 的可靠性增加时，其与 t_3 的重要性比值会不断下降，这反映了较低可靠性的组件会拥有较高的相对重要性的事实，因为这种组件比较容易成为整个系统的瓶颈；图 6 则显示了在 t_2 的执行次数增加时，其与 t_3 的重要性比值会不断上升，这就进一步验证了组件重要性与其执行次数的关系. 从这两张图可以看出，我们的组件重要性这一评估结果是综合考虑了组件可靠性和其运行次数这两个方面的.

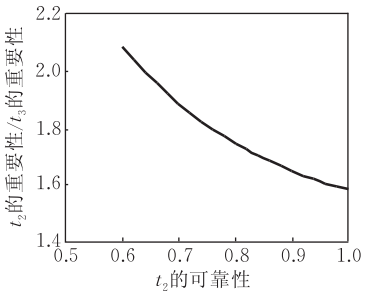

图 5 组件可靠性对其重要性的影响

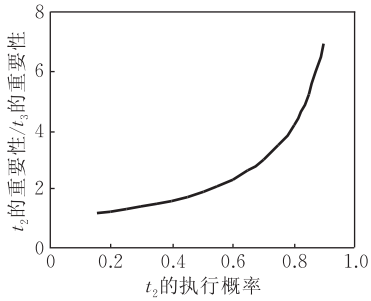

图 6 组件运行次数对其重要性的影响

6 总结与展望

现有软件系统的运行环境逐渐朝着更加开放的方向发展，在这种情况下如何对软件的可靠性进行快速有效的衡量成为一个急需解决的问题. 本文首先对传统软件可靠性评估方法的应用范围和效率等方面在新形势下的不足进行了分析，然后提出了利用 Petri 网来作为软件体系结构描述的工具，并对传统的基于状态的可靠性评估方法加以改进的思路，最终得到了一个能处理具有各种控制结构的软件系统的方法，该方法不但能快速并准确地得到评估结果，同时还能产生对组件重要性的衡量，来为系统的分析和优化提供更多的参考.

在今后的工作中，还需要通过实际项目中的应用来进一步检验该方法的有效性和可操作性；同时，还需要研究如何获取软件运行行为的动态信息，从而对系统的评估结果进行实时更新，使其能够及时地反映组件和体系结构的变化；此外，本文给出了组件重要性这一评估结果，如何利用其来对系统进行优化和改进也是一个可以深入研究的方向.

参 考 文 献

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

[2] Lu Jian, Ma Xiao-Xing, Tao Xian-Ping, Xu Feng, Hu Hao. Research and progress on internetware. Science in China (Series E), 2006, 36(10): 1037-1080(in Chinese)
(吕建，马晓星，陶先平，徐锋，胡昊. 网构软件的研究与进展. 中国科学 E 辑，2006，36(10)：1037-1080)

[3] Ran Shuping. A model for Web services discovery with QoS. ACM SIGecom Exchanges, 2003, 4(1): 1-10

[4] Sahai Akhil, Machiraju Vijay, Sayal Mehmet, van Moorsel Aad P A, Casati Fabio. Automated SLA monitoring for Web services//Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications, 2002: 28-41

[5] Liu W. Trustworthy service selection and composition-reducing the entropy of service-oriented web//Proceedings of the 3rd IEEE International Conference on Industrial Informatics (INDIN). Perth, Australia, 2005: 104-109

[6] Goseva-Popstojanova K, Mathur A P, Trivedi K S. Comparison of architecture-based software reliability models//Proceedings of the 12th IEEE International Symposium on Software Reliability Engineering (ISSRE-2001). Hong Kong, China, 2001: 22-31

[7] Gokhale Swapna S, Trivedi Kishor S, Wong W Eric, Horgan J R. An analytical approach to architecutre-based software reliability prediction//Proceedings of the IEEE International Computer Performance and Dependability Symposium. Dallas, 1998: 13-22

[8] Wang Wen-Li, Wu Ye, Chen Mei-Hwa. An architecture-based software reliability model//Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing. Hong Kong, China, 1999: 143-150

[9] Dolbec Jean, Shepard Terry. A component based software reliability model//Proceedings of the 1995 Conference of the Centre for Advanced Studies on Collaborative Research. Toronto, Ontario, Canada, 1995: 19

[10] Gokhale S S. Architecture-based software reliability analysis: Overview and limitations. IEEE Transactions on Dependable and Secure Computing, 2007, 4(1): 32-40

[11] Goel A L, Okumoto K. Time-dependent error-detection rate models for software reliability and other performance measures. IEEE Transactions on Reliability, 1979, 28(3): 206-211

[12] Musa J D, Okumoto K. Logarithmic poisson execution time model for software reliability measurement//Proceedings of the COMPSAC. Chicago, 1984: 230-238

[13] Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs, NJ: Prentice Hall, 1996

[14] Parnas D L. The influence of software structure on reliability//Proceedings of the 1975 International Conference Reliable Software. Los Angeles, CA, 1975: 358-362

[15] Shooman M L. Structural models for software reliability prediction//Proceedings of the 2nd International Conference Software Engineering. San Fransisco, CA, 1976: 268-280

[16] Martin D, Burstein M, Hobbs J et al. OWL-S: Semantic markup for Web services. W3C Member Submission, Nov. 22, 2004

[17] van der Aalst W M P. The application of Petri nets to workflow management. The Journal of Circuits, Systems and Computers, 1998, 8(1): 21-66

[18] Andonoff Eric, Bouzguenda Lotfi, Hanachi Chihab. Specifying workflow Web services using Petri nets with objects and generating of their OWL-S specifications//Proceedings of the 6th International Conference on Electronic Commerce and Web Technologies (EC-Web'05). Copenhagen, Denmark, 2005: 41-52

[19] Zhang Guang-Mei, Li Xiao-Wei. Early prediction of software reliability parameter based on software architecture. Computer Engineering, 2005, 31(5): 90-92(in Chinese)
(张广梅，李晓维. 基于体系结构的软件可靠性参数早期预测. 计算机工程，2005，31(5)：90-92)

[20] Gokhale S. Quantifying the variance in application reliability//Proceedings of the Pacific Rim Dependability Conference. Papeete, Polynesia, 2004: 113-121

[21] Siegrist K. Reliability of systems with Markov transfer of control. IEEE Transactions on Reliability, 1988, 14(7): 1049-1053

LU Wen, born in 1984, M. S. candidate. His research interests include system security and trusted computing.

XU Feng, born in 1975, Ph. D., professor. His research interests include trust management, reputation system, trusted computing, software reliability.

LV Jian, born in 1960, Ph. D., professor, Ph. D. supervisor. His research interests include software methodologies, software automation, software agents, and middleware systems.

Background

Along with the rapid development of Internet, developers now begin to compose components scattered all over the Internet to build more complicated systems. It's meaningful and urgent to find a method to evaluate the reliability of this new kind of software system. However, existing methods have flaws on some aspects. Most of them merely evaluate the components, and are unable to check the reliability of the whole system composed by them, while others can monitor the whole system, but still have problems in some situations. Firstly, they can hardly deal with concurrency, which is inevitable and common in an open environment. Secondly, open environment requires far more efficiency than ever before; existing methods are generally based on a less-efficient thorough analysis. Also, they are weak in the measurement on the importance of the components, which is useful and necessary in software analysis, such as optimizations.

To solve these problems, this paper put forward an improved reliability evaluation method which can do better in both efficiency and applicability. It first described how to use Petri nets as software architecture description; and then introduced a bottom-up way to calculate the system's reliability, the calculation process could be done efficiently and accurately, and could also deal with systems with concurrency, which can hardly be handled by existing methods; the method could also generate information reflecting the component's importance to the whole system, which can make software reliability evaluation play a more important role in software development.

This paper is supported by the National Basic Research 973 Program of China under grant No. 2009CB320702, the National Hi-Tech Research and Development 863 Program of China under Grant Nos. 2007AA01Z178, 2007AA01Z140 and 2009AA01Z114, and the National Natural Science Foundation of China under grant Nos. 60603034, and the JSNSF No. BK2008017.

The project is to make a research on building high confidence software systems in the open environment with the help of the methods in trust computing. Several papers have already been published under this project, mainly on the trust evaluation and evolvement, including a semantic similarity based time related recommendation-feedback trust model, a trust evolution model for P2P networks, and so on.

With the former work which is mainly on the evaluation and management of the single software components, the work done in this paper can further make an evaluation on the whole software component, and generate more useful and meaningful results.