基于构件的NHPP类软件可靠性增长模型的研究

侯春燕 崔 刚 刘宏伟 杨孝宗
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

摘 要 随着基于构件的软件开发模式的迅速发展,传统的NHPP模型无法适应大型的基于软构件的新型软件开发模式。结合软件可靠性分析中的黑盒方法和白盒方法,提出一种基于构件的NHPP类软件可靠性增长模型,CB-NHPP模型。该模型以可加模型为基础,实现了时间域模型和体系结构域模型的结合,克服了这两种技术无法同时考虑软件测试过程中的故障排除和软件体系结构的问题。由于同时考虑了更多因素,因此该模型具有更高的准确性。最后通过实验证明了CB-NHPP模型的有效性。

关键词 非齐次泊松过程,软件可靠性增长模型,可加模型,构件,基于体系结构的软件可靠性
中图法分类号 TP311 **文献标识码** A

Study of Component-based NHPP Software Reliability Model

HOU Chun-yan CUI Gang LIU Hong-wei YANG Xiao-zong
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract With the rapid development in component-based software development patterns, traditional NHPP models can't adapt themselves to the new large component-based software development patterns. A component-based software reliability growth model, CB-NHPP model, was proposed by combining black-box methods with white-box methods in software reliability analysis. Based on additive model, CB-NHPP model integrates time domain models with structure domain models to simultaneously address system structures and software with repair. This model is more precise because it considers more factors. At last an example is evaluated to validate and show the effectiveness of CB-NHPP model.

Keywords Non-homogeneous poisson process, Software reliability growth model, Additive model, Components, Architecture-based software reliability

1 引言

随着计算机应用领域的不断拓展,软件的规模越来越大,结构和功能越来越复杂,人们对高质量软件的需求也更加迫切。软件可靠性是软件质量的重要指标之一,在过去几十年中人们对其进行了大量的研究,这些研究工作主要集中在对非齐次泊松过程(NHPP)类软件可靠性增长模型的研究上[1-3]。到目前为止,已经提出了不下于上百种NHPP类软件可靠性增长模型。

NHPP模型是时间域模型,建模软件可靠性为时间的函数。它是一种黑盒模型,把软件看成一个单调的整体,只考虑软件同外部环境的交互,而不考虑软件内部结构。目前,随着软件产业的快速发展,软件迅速脱离"一切从零开始"的开发模型,转向高级复用技术。黑盒模型明显不能适应大型的基于软构件的新型软件开发模型,于是关于软件可靠性系统分析的方法得到重视,成为软件系统可靠性研究的主导方法之一[4]。根据软构件的第三方开发和基于构件的软件开发特点,该方法虽然具体的软构件仍是黑盒,但要求系统内部的结构信息可见。围绕该方法提出了一系列基于结构的软件可靠性估计模型,即白盒模型[5]。

在实际软件开发中,不同构件具有不同的失效强度、最后失效发生时间和测试时间,因此基于结构的方法采用白盒模型来考虑不同构件的特征。白盒模型能够解决软件构件之间的控制转移。与黑盒模型相比,白盒模型得到的软件可靠性测度是软件结构的函数,是一个单一估计,而不是一系列随时间变化的预测值。白盒模型的这种"定常(time -invariant)"特性无法解决软件测试过程中的故障排除问题,这使得它们还不具有实际应用价值[6]。

本文从基于软构件的软件测试的实际情况出发,提出一种基于构件的NHPP类软件可靠性增长模型,即CB-NHPP模型。该模型同时考虑了软件可靠性增长和软件体系结构,实现了黑盒模型和白盒模型的结合,也就是时间域模型和结构域模型的结合。CB-NHPP模型在传统NHPP模型的基础上考虑软件体系结构,能够真正满足基于构件的新型软件可

到稿日期:2008-11-20 本文受国家自然科学基金项目(60503015),教育部博士点基金(20020213017)资助。
侯春燕(1980-),女,博士研究生,主要研究方向为软件测试、软件可靠性评估、容错计算,E-mail:houchunyan@ftcl.hit.edu.cn;崔 刚(1949-),男,教授,博士生导师,主要研究方向为容错计算、可穿戴计算、软件可靠性评估;刘宏伟(1971-),男,副教授,主要研究方向为软件测试、软件可靠性评估、可穿戴计算;杨孝宗(1939-),男,教授,博士生导师,主要研究方向为可穿戴计算、容错计算、软件测试。

靠性估计的需要。

2 NHPP 类软件可靠性增长模型

NHPP 类软件可靠性增长模型是软件可靠性增长模型中非常重要的一类，并且已经成为软件可靠性工程实践中非常重要的工具[7]，是管理和提高软件可靠性过程中最有吸引力的一类模型[8]。

注释：

$N(t)$：$t\geqslant 0$，一个计数过程，表示到时刻 t 检测到的累计故障数；

a：软件系统中潜伏的故障总数的初始值；

$a(t)$：基于时间的潜伏故障总数的函数，即到时刻 t 已排除的软件故障数和潜伏在软件中尚未被发现的软件故障数的和；

b：故障检测率的初始值。故障检测率表示软件内残存一个差错的差错发现率，故障检测率的大小可以说明任意时刻下发现软件内残存差错的难易度；

$b(t)$：基于时间的故障检测率，即软件中每个故障在时刻 t 被检测到的平均概率；

$\lambda(t)$：失效强度函数，即在时刻 t，软件单位时间内的失效数；

$m(t)$：到时刻 t 为止能够发现的故障数的期望值；

$R(x|t)$：可靠度函数，即 $P\{$系统在$(t,t+x)$时间内不发生失效$|$最近一次失效发生在时刻 $t\}$。

NHPP 类软件可靠性增长模型用一个计数过程 $\{N(t),t\geqslant 0\}$ 表示到时刻 t 为止检测到的软件故障累计数。如果故障累计数的期望值函数用 $m(t)$ 表示，则一个基于 NHPP 过程的软件可靠性增长模型的一般形式如下所示：

$$P\{N(t)=n\}=\frac{m(t)^n}{n!}\exp(-m(t)) \tag{1}$$

失效强度函数为

$$\lambda(t)=\frac{\mathrm{d}m(t)}{\mathrm{d}t} \tag{2}$$

如果最后一次失效发生在测试时刻 t，则被测软件在时间段$(t,t+s)$内可靠度为

$$R(x|t)=\exp[-(m(t+x)-m(t))] \tag{3}$$

故障检测率可表示为

$$b(t)=\frac{\lambda(t)}{a(t)-m(t)} \tag{4}$$

G-O 模型于 1979 年由 Goel 和 Okumoto 提出[9]，是关于连续时间的 NHPP 模型中的经典模型。在很多具体的应用中，G-O 模型都工作得很好[10]。G-O 模型假设故障检测率在测试过程中保持不变，被发现的故障立即被排除，排除故障时不引入新的故障。因此：

$$m(t)=a(1-\exp(-bt)) \tag{5}$$

3 基于结构的软件可靠性模型

从 20 世纪 90 年代中期到末期开始，随着软件大小和复杂度的增加，基于体系结构的软件可靠性分析开始受到越来越多的关注[11]。基于体系结构的软件可靠性分析的主要目的是基于构件可靠性和应用体系结构得到软件的可靠性估计。基于体系结构的软件可靠性分析具有很多优点，它能够根据软件的体系结构和每个构件的可靠性得到整个软件的可靠性，能够分析软件可靠性相对于构件可靠性和应用体系结构的灵敏度，能够识别可靠性瓶颈，能够更早地估计软件的可靠性。

但是目前提出的基于体系结构的软件可靠性分析方法具有很大的局限性，存在一些问题需要解决。其中一个关键问题就是基于体系结构的软件可靠性分析得到的软件可靠性测度是软件结构的函数，是一个单一估计，而不是一系列随时间变化的预测值。这种“定常(time -invariant)”特性无法解决软件测试过程中的故障排除问题[6]，这使得基于体系结构的软件可靠性分析方法还不具有实际应用价值。而且已经提出的方法中，绝大多数方法都假设构件可靠性已知，而不考虑如何确定构件的可靠性。

一些研究工作[6,12-15]考虑将黑盒模型和白盒模型相结合，也就是将软件可靠性增长模型应用到基于体系结构的软件可靠性分析中。这些工作中大部分[6,12,13]只是应用软件可靠性增长模型来估计每个构件的可靠性，但是没有改变估计结果的“定常(time -invariant)”特性，没有建立整个软件系统的可靠性增长模型。

文献[14,15]中提出了可加模型(additive model)，从黑盒模型和白盒模型结合的角度出发，建立了基于构件的软件可靠性增长模型。基于体系结构的软件可靠性模型分为 3 类：基于状态的模型、基于路径的模型和可加模型。前两类模型提出较早，属于纯白盒模型。可加模型考虑到白盒模型的局限性，将黑盒模型应用到基于体系结构的软件可靠性分析中。目前已经提出的可加模型有两种：X-W 模型[14]和 Everett 模型[15]。称为可加模型是因为模型假设，用 NHPP 建模构件可靠性，系统失效强度等于构件失效强度的总和。

可加模型假设系统由 n 个构件组成，所有构件并行开发，独立测试。假设构件失效为系统失效，系统为串行系统。如果用 NHPP 建模构件可靠性，构件失效强度为 $\lambda_i(t)$，那么系统失效强度为

$$\lambda_s(t)=\lambda_1(t)+\lambda_2(t)+\cdots+\lambda_n(t) \tag{6}$$

到时刻 t 系统累计失效数的期望值为

$$m_s(t)=\sum_{i=1}^{n}m_i(t)=\int_0^t\sum_{i=0}^{n}\lambda_i(\tau)\mathrm{d}\tau \tag{7}$$

可加模型是软件可靠性分析领域中的创新性尝试，它根据基于构件的软件开发的实际特点，将软件可靠性分析中的两类模型，黑盒模型和白盒模型结合起来，取长补短，同时解决两类模型存在的问题，使软件模型适应基于构件的软件可靠性分析的实际需要，引起软件可靠性领域的广泛关注。但是可加模型仍存在如下问题：

(1)可加模型没有明确考虑软件的体系结构。它们没有实现对软件体系结构建模，只是利用构件的失效数据来估计整个软件的可靠性。

(2)可加模型没有考虑不同构件的测试特征。在对每个构件执行的单元测试中，根据每个构件的具体特征，对其执行的测试是不同的。

4 CB-NHPP

基于可加模型，我们提出一种新的基于构件的 NHPP 类软件可靠性增长模型，即 CB-NHPP 模型。CB-NHPP 模型实现了对可加模型的改进，解决了可加模型所存在的问题。

可加模型存在的最大问题就是没有明确考虑软件的体系结构，没有实现对软件体系结构建模。我们结合基于状态的软件可靠性分析方法，解决对软件体系结构建模的问题。

在基于体系结构的软件可靠性分析方法中，基于状态的方法提出最早，而且基于体系结构的软件可靠性分析中大部分工作都集中在对基于状态的方法的研究上。与基于路径的方法相比，基于状态的方法具有如下优势：

(1)基于状态的方法可以解决由于环路存在而具有无数条路径的情况。

(2)基于状态的方法根据可用信息的级别，可以采用不同类型的构件失效模型。因此，基于状态的方法可以用于整个软件开发生命周期中。

(3)在基于状态的方法中，等级方法生成的可靠性分析函数可以实现更早的灵敏度分析和预测分析。

基于状态的模型属于纯白盒模型，它们明确地考虑了软件体系结构，实现对各种软件应用的体系结构建模。软件体系结构表示组成软件的不同构件之间相互交互的方式。体系结构也包括每个构件的执行时间信息(均值、方差、分布)。基于状态的方法中，软件体系结构建模为 DTMC(离散时间马尔科夫链)，CTMC(连续时间马尔科夫链)，SMP(半马尔科夫过程)。它们可进一步分为吸收和不可约的马尔科夫链。前者表示按照用户需求运行的软件，每次运行都清楚地对应一个终止状态。后者表示持续运行的软件应用，例如实时控制系统[5]。

基于结构的软件可靠性分析中一个关键问题是如何考虑每个构件的使用情况。对于不同类型的软件这个问题的解决方式不同。本文考虑可终止的软件应用，其体系结构建模为吸收 DTMC。DTMC 可以表示为它的一步概率转移矩阵，$\vec{P}=[p_{ij}]$，其元素是转移概率。

定义向量 $\vec{\eta}=[\eta_i]$ 表示极限状态向量。如果有限 DTMC 是遍历的，它的稳定状态分布可由下列矩阵方程进行评价：

$$\vec{\eta}=\vec{\eta}\vec{P} \tag{8}$$

我们用构件执行时间比例作为测度来衡量构件的使用情况。设 π_i 表示构件 i 的执行时间比例，定义 π_i 为

$$\pi_i=\frac{\eta_i\tau_i}{\sum_{i=1}^{n}\eta_i\tau_i} \tag{9}$$

其中 τ_i 表示构件 i 一次执行的平均执行时间。

以上分析实现了对软件体系结构的明确考虑。在此基础上，我们介绍 CB-NHPP 模型。对于该模型做出如下假设：

(1)软件系统由 n 个构件组成，所有构件并行开发，独立测试；

(2)胶合逻辑完全可靠。较大规模的胶合逻辑可以抽象为独立的构件参加到软件产品中，剩下较小规模的胶合逻辑则假设足够简单；

(3)起始时间为集成测试开始的时间；

(4)软件系统到时刻 t 累计发生失效数等于所有构件在各自所占有的运行时间内累计发生失效数的总和。

用 NHPP 建模构件可靠性，到时刻 t 构件 i 累计失效数的期望值表示为 $m_i(t)$。则根据以上假设，到时刻 t 软件系统累计失效数的期望值为

$$m_s(t)=\sum_{i=1}^{n}[m_i(\pi_i t+T_i)-m_i(T_i)] \tag{10}$$

其中 π_i 表示构件 i 的执行时间比例，如式(9)。则 $\pi_i t$ 表示到时刻 t 构件 i 的累计执行时间。对于硬件系统，认为所有构件处于持续运行状态，则相对应所有 π_i 都等于 1。参数 π_i 实现了对软件体系结构的明确考虑。T_i 表示单元测试时构件 i 最后一次失效发生的时间，则 $m_i(T_i)$ 表示构件 i 单元测试期间累计失效数的期望值。参数 T_i 实现了对不同构件的测试特征的考虑。

式(10)中，不同构件可以根据其单元测试的实际情况用不同的 NHPP 模型进行建模。这里为了讨论方便，对于每个构件用 G-O 模型建模。

根据式(5)，到时刻 t 构件 i 累计失效数的期望值为

$$m_i(t)=a_i(1-\exp(-b_i t)) \tag{11}$$

其中 a_i 表示构件 i 中潜伏的故障总数，b_i 表示构件 i 的故障检测率。

由式(10)和式(11)，可以得出

$$m_s(t)=(\sum_{i=1}^{n}a_i\exp(-T_i b_i))(1-\exp(-(\sum_{i=1}^{n}\pi_i b_i)t)) \tag{12}$$

由上式可以看出，得到的整个软件系统可靠性模型也为 NHPP 模型。其失效强度为

$$\lambda_s(t)=(\sum_{i=1}^{n}a_i\exp(-T_i b_i))(\sum_{i=1}^{n}\pi_i b_i)\exp(-(\sum_{i=1}^{n}\pi_i b_i)t) \tag{13}$$

软件系统的可靠性为

$$R(t)=\exp(-\int_0^t\lambda_s(\tau)\mathrm{d}\tau)=\exp(-(\sum_{i=1}^{n}a_i\exp(-T_i b_i))(1-\exp(-(\sum_{i=1}^{n}\pi_i b_i)t))) \tag{14}$$

5 实例分析

为了对 CB-NHPP 模型的有效性进行验证，我们对一个信息管理系统的 1 个子系统进行测试。该信息管理系统用来对专利事务流程进行管理，测试的子系统管理其中 1 个子流程。该子系统由 5 个构件组成，用 DELPHI 语言开发，共有 40,372 行代码，362 个函数。目前该子系统已经发布使用。

根据被测试子系统的操作剖面，建立其体系结构图如图 1 所示。有向边表示一个构件与另一个构件之间的交互。每次执行从构件 c_1 开始，执行到构件 c_5 结束。该系统的 CB-NHPP 模型的建立过程可以分为以下 3 个步骤。

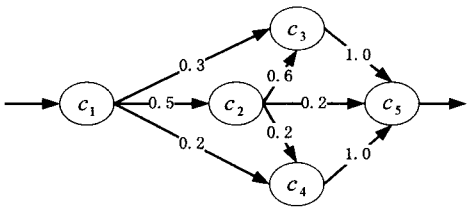

图 1 由 5 个构件组成的软件应用结构

首先，对软件体系结构建模，得到构件执行时间比例测度。用 DTMC 对体系结构建模，每个构件的平均执行时间如表 1 所列，根据式(9)，得出每个构件执行时间比例为：

$$\vec{\pi}=[0.051\quad 0.408\quad 0.301\quad 0.122\quad 0.118]$$

表 1 5 个构件的测试数据

构件	平均执行时间(毫秒)	构件失效发生时间(秒)

c_1	4068	341, 1737, 2519, 6542, 7636, 10988, 12350, 13964, 15860, 18062, 25641, 28698, 31382, 36922, 40827, 45703, 58444, 66765
c_2	65294	166, 2081, 3551, 5281, 5693, 5878, 7148, 8772, 9907, 12904, 13680, 14943, 18026, 19071, 24092, 25344, 27529, 36592, 41942, 49840, 53564, 76183, 82502
c_3	40157	985, 2955, 5720, 6393, 9284, 10423, 14622, 15012, 16629, 20523, 25758, 27421, 30669, 37654, 45797, 49697, 59375, 78621
c_4	32652	1126, 3409, 7895, 8724, 10531, 12492, 16738, 20332, 25581, 28229, 36584, 38746, 42721, 46218, 69075
c_5	9467	230, 1087, 5474, 8439, 10735, 16492, 21147, 27409, 30855, 39361, 40945, 48634, 54816, 60762, 73819

然后，根据构件单元测试数据，建立每个构件的 NHPP 模型。在单元测试中，根据不同构件的重要程度，测试中发生失效数目，测试的执行时间不同。由于保密的原因，我们对数据进行规格化处理，得到处理后的构件测试数据如表 1 所列。用 G-O 模型对每个构件进行建模，通过最小二乘法对参数进行估计，估计值如表 2 所列。

表 2　构件 G-O 模型的参数

构件	a	b
c_1	18.4726	4.0506e-005
c_2	22.6537	4.2226e-005
c_3	19.3413	3.9341e-005
c_4	17.4642	4.0717e-005
c_5	17.4893	3.4075e-005

最后，基于构件的 NHPP 模型以及执行时间比例，根据式(12)，建立整个软件系统的 CB-NHPP 模型。为了进行比较，我们也分别建立了该软件系统的可加模型和 G-O 模型，如图 2 所示。

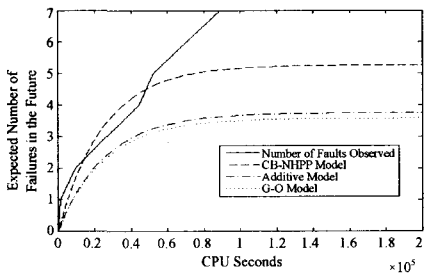

图 2　集成后软件系统失效数的期望值

为了评测 CB-NHPP 模型的性能，可用误差平方和(Sum of Squared Errors，SSE)度量曲线拟合效果。

$$SSE=\sum_{i=1}^{n}(y_i-\overline{m}(t_i))^2 \tag{15}$$

其中 n 表示失效数据集中失效样本的数量，$\overline{m}(t_i)$ 表示到 t_i 时刻为止故障累计数的估算值，y_i 表示到 t_i 时刻为止故障累计数的实测值。

SSE 的值越小，曲线拟合得越好。

为了检验 CB-NHPP 模型的拟合效果，在集成测试数据(如表 3)上分别应用 CB-NHPP 模型、G-O 模型、可加模型进行曲线拟合，拟合结果如表 4 所示。从图 2 以及拟合结果可以看出，CB-NHPP 模型的拟合效果优于其他两种模型。

表 3　集成测试的测试数据

累计失效数	失效发生时间(秒)
1	1143
2	9286
3	26467
4	44319
5	52150
6	70359
7	89246

表 4　模型的数据拟合情况对比

比较标准	CB-NHPP 模型	可加模型	G-O 模型
SSE	5.8067	23.1771	25.9388

结束语　本文提出一种新的基于构件的 NHPP 类软件可靠性增长模型——CB-NHPP 模型。该模型综合了黑盒模型和白盒模型的优点，同时考虑了软件体系结构和软件测试过程中的故障排除。由于 CB-NHPP 模型同时考虑了更多因素，因此具有更高的准确性。CB-NHPP 模型可以实现对集成后软件系统可能发生失效数的预测，这有助于确定整个系统的成熟度。最后以一个实例比较了经典的 G-O 模型、可加模型和 CB-NHPP 模型，结果表明 CB-NHPP 模型的拟合情况比 G-O 模型和可加模型的好。

参 考 文 献

[1] Pham H, Zhang X. NHPP Software Reliability and Cost Model with Testing Coverage. Europe Journal of Operational Research, 2003, 145(3): 443-454

[2] Huang C Y, Kuo S Y, Lyu M R. An Assessment of Testing-Effort Dependent Software Reliability Growth Models. IEEE Trans. on Reliability, 2007, 56(2): 198-211

[3] Huang C Y, Lyu M R, Kuo S-Y. A unified scheme of some non-homogeneous Poisson process models for software reliability estimation. IEEE Trans. Softw. Eng., 2003, 29(3): 261-269

[4] 毛晓光，邓勇进. 基于构件软件的可靠性通用模型[J]. 软件学报，2004，15(1)：27-32

[5] Gŏseva-Popstojanova K, Trivedi K S. Architecture-based approach to reliability assessment of software systems. Performance Evaluation, 2001, 45(2/3): 179-204

[6] Wang Wen-li, Hemminger T L, Tang M-H. A Moving Average Non-Homogeneous Poisson Process Reliability Growth Model to Account for Software with Repair and System Structures. IEEE Trans. Reliability, 2007, 56(3): 411-421

[7] 陈松林，鲁星，马妙技. 高技术武器系统中软件的现状及应用分析[J]. 航空兵器，2003，21(3)：33-35

[8] HO S L, Xie M, GOH T N. A Study of the Connectionist Models for Software Reliability Prediction. Computer and Mathematics with Applications, 2003, 46(7): 1037-1045

[9] 刘宏伟. 非齐次泊松过程类软件可靠性增长模型研究[D]. 哈尔滨：哈尔滨工业大学，2004

[10] Pham H, Zhang X. A software cost model with warranty and risk costs. IEEE Trans. Computers, 1999, 48(1): 71-75

[11] Gokhale S S. Architecture-based Software Reliability Analysis: Overview and Limitations. IEEE Trans. on Dependable and Secure Computing, 2007, 4(1): 32-40

[12] Wang W-L, Hemminger T L, Tang M-H. A moving average Modeling Approach for Computing Component-based Software

reliability growth trends. INFOCOMP Journal of Computer Science, 2006, 5(3): 9-18

[13] Gokhale S, Wong W E, Trivedi K S, et al. An Analytic Approach to Architecture-based Software Performance and Reliability Prediction. Performance Evaluation, 2004, 58(4): 391-412

[14] Xie M, Wohlin C. An additive reliability model for the analysis of modular software failure data // Proceedings of the Sixth International Symposium on Software Reliability Engineering (ISSRE'95). 1995: 188-194

[15] Everett W. Software component reliability analysis // Proceedings of the Symposium on Application-specific Systems and Software Engineering Technology (ASSET'99). 1999: 204-211

(上接第 191 页)

讨论过，本文按照前述假设作相应调整，具体如式(4)所示：

$$Q_{et(ret)}=\frac{e^{\lambda T_D}(e^{\lambda T_F}-1)}{\lambda} \tag{4}$$

对于某种错误处理服务的执行费用按照资源数目、资源价格以及执行时间 3 个方面来计算。重试错误处理服务的执行费用按照式(5)来计算。

$$Q_{co}=Q_{price} * Q_{et(ret)} \tag{5}$$

2.2.2 检查点

Vaidya 最早在文献[8]中讨论了检查点错误处理技术下任务的执行时间，本文根据前述定义与假设，按照概率相关理论，网格上检查点错误处理服务的执行时间如式(6)所示。

$$Q_{et(ch)}=\left(\frac{T_F}{T_I}\right)\frac{e^{\lambda(T_D-T_C+T_R)}(e^{\lambda(T_I+T_C)}-1)}{\lambda} \tag{6}$$

与重试错误处理服务类似，检查点错误处理服务的执行费用可以按照式(7)计算：

$$Q_{co}=Q_{price} * Q_{et(ch)} \tag{7}$$

2.2.3 带有重试的副本

按照前面的分析，带有重试的副本错误处理方法的任务执行时间可以按照式(8)来计算。

$$Q_{et(ret+rep)}=\frac{e^{N_R^{\lambda}T_D}(e^{N_R^{\lambda}T_F}-1)}{\lambda/N_R} \tag{8}$$

带有重试的副本处理方法的执行代价是：

$$Q_{co}=Q_{price} * Q_{et(rep+rep)} * N_R \tag{9}$$

3 基于服务质量的通用决策模型

3.1 模型的建立

根据多属性决策方法的思想，基于 QoS 的错误处理服务选择问题可抽象为如式(10)所示的决策模型：

$$FHS_{QoS}=\{A, Dm, Cs\} \tag{10}$$

其中，$A=[a_{ij}]_{m\times n}$是服务质量决策矩阵，$a_{ij}=Q_i(fhs_j)$是服务$fhs_j\in FHS$在服务质量属性$Q_i\in QoS$的取值，这里$QoS=\{Q_1, Q_2, \cdots, Q_n\}$表示服务质量属性集，$FHS=\{fhs_1, fhs_2, \cdots, fhs_m\}$表示错误处理服务的集合；$D_m=\{dm_1, dm_2, \cdots, dm_u\}$表示决策模式集，如一个包含单赋权模式、主观赋权模式、客观赋权模式和主客观赋权模式的集合记为：$D_m=\{$*single weight mode, subjective weight mode, object weight mode, subjective-objective weight mode*$\}$；

$Cs=\{cs_1, cs_2, \cdots, cs_v\}$表示对决策矩阵的约束集，如对不同服务属性设定一定的阈值。

3.2 决策矩阵的标准化

为了正确评价各种错误处理服务，决策矩阵 A 需要标准化，本质是给出某个属性值在决策评价优劣时的实际价值。通过下列公式获得 $A=[a_{ij}]_{m\times n}$ 标准化后对应矩阵 $B=[b_{ij}]_{m\times n}$：

· 效益型属性(Benefit Property)

$$b_{ij}=\begin{cases}\dfrac{a_{ij}-a_j^{\min}}{a_j^{\max}-a_j^{\min}} & a_j^{\max}\neq a_j^{\min}\\ 1 & a_j^{\max}=a_j^{\min}\end{cases} \tag{11}$$

其中，$a_j^{\max}=\max_i a_{ij}$，$a_j^{\min}=\min_i a_{ij}$，$i=1,2,\cdots,m$；$j=1,2,\cdots,n$。

· 成本型属性(Cost Property)

$$b_{ij}=\begin{cases}\dfrac{a_j^{\max}-a_{ij}}{a_j^{\max}-a_j^{\min}} & a_j^{\max}\neq a_j^{\min}\\ 1 & a_j^{\max}=a_j^{\min}\end{cases} \tag{12}$$

其中，$a_j^{\max}=\max_i a_{ij}$，$a_j^{\min}=\min_i a_{ij}$，$i=1,2,\cdots,m$；$j=1,2,\cdots,n$。

由属性值的标准化后，每个属性值的最优值都是 1 且最差值为 0。

结束语 本文根据网格计算环境下的错误处理需求，给出了错误处理服务的服务质量标准，建立了相应的服务质量模型；将基于 QoS 的动态错误处理服务选择问题抽象为多属性决策问题，建立了相应的决策模型。下一步的工作，将在本文的基础上，提出基于服务质量的错误处理服务选择算法，并通过仿真实验验证模型及算法的正确性和有效性。

参 考 文 献

[1] Nguyen-Tuong A. Integrating Fault-Tolerance Techniques in Grid Applications[D]. Ph. D Dissertation. University of Verginia, 2000

[2] Hwang S. Grid workflow: a flexible framework for fault tolerance in the grid[D]. Ph. D Dissertation. University of Southern California, 2003

[3] 石宣化. 可信赖网格理论与关键技术研究[D]. 武汉：华中科技大学，2005，9

[4] Zeng Liangzhao, Benatallah B, Dumans M, et al. Quality driven web services composition[C]// Proc. of 12th International Conference on World Wide Web (WWW-03). Budapest, Hungary, 2003

[5] Sheth A, Cardoso J, Miller J, et al. QoS for service-oriented middleware[C]// Proc. of the 6th World Multiconference on Systemics, Cybernetics and Informations (SCI02). Orlndo, FL, 2002

[6] Liu Yutu, Ngu A H H, Zeng Liangzhao. QoS computation and policing in dynamic web service selection[C]// Proc. of 13th International Conference on World Wide Web (WWW-04). New York, USA, 2004

[7] Duda A. The effects of checkpointing on program execution time [J]. Information Processing Letters, 1983, 16: 221-229

[8] Vaidya N H. Impact of checkpoint latency on overhead ratio of a checkpointing scheme[J]. IEEE Transactions on Computers, 1997, 46(8): 942-947

[9] Hwang C L, Yoon K. Multiple attribute decision making[M]. Berlin Heidelberg New York: Springer-Verlag, 1981

[10] Nguyen-Tuong A. Integrating Fault-Tolerance Techniques in Grid Applications[D]. Ph. D Dissertation. University of Verginia, 2000