国产基础软件可靠性分析及验证①

阚丹丹

(江苏自动化研究所, 连云港 222061)

摘 要: 国产基础软件通过常规测试后仍存在诸多质量或可靠性问题, 为此, 针对国产基础软件的特性及现有可靠性测试技术的不适应之处, 结合 Musa 剖面模型和 Markov 链模型设计出一种带标记的 Markov 链运行剖面建模技术来构建操作系统的运行剖面, 研究并给出相应的可靠性测试用例生成方法, 最后选取典型操作系统进行了分析验证, 证明了研究成果的有效性.

关键词: 软件测试; 国产; 基础软件; 剖面模型; 可靠性测试

Reliability Analysis and Testing for Domestic Foundational Software

KAN Dan-Dan

(Jiangsu Automation Research Institute, Lianyungang 222061, China)

Abstract: After routing testing, domestic foundational software still has a few reliability problems. As a result, based on the characteristic of domestic foundational software and its special requirement on reliability, the paper combines with Musa profile model and Markov chain model and proposes a Markov chain model with tags. A method of generating the reliability testing cases is presented. Finally, we analyze and validate the reliability testing method proposed here and the research is provided to be useful.

Key words: software testing; domestic; foundational software; profile model; software reliability testing

近年来, 我国对于信息系统的安全性、可靠性的重视程度日益提升, 作为信息系统基石的基础软硬件发展更是成为焦点, 目前在国产处理器、国产操作系统领域都投入了大量的人力、物力, 取得了一系列显著的成果. 目前, 国内已有几家单位相继研制出了具有自主知识产权的处理器(方舟、龙芯等)以及国产操作系统(中标麒麟等)、数据库(人大金仓、神舟通用等)和中间件(东方通中间件、金蝶中间件等)等基础软件, 并逐渐应用到通信、军事、航空、航天等高精尖技术及实时性要求极高的领域. 但是在国产基础软硬件迅速发展的过程中, 也伴生了很多的问题, 其中产品质量是其中一个重要方面. 基础软硬件的体系比较复杂, 并且相互之间的结合比较密切, 包含关键技术多, 其开发过程本身是一个复杂的系统工程. 国外基础软硬件的发展的底蕴比较深厚, 先后历经几代数十年. 所以, 对国产基础软件可靠性分析及验证技术进行深入探讨和研究就非常有必要.

目前, 针对国产基础软件测试技术的研究有很多, 但是仍然无法保证国产化基础软件的可靠性需求. 首先, 国产化基础软件的指令集差异导致国外基准测试用例集不完全适应于国产化基础软件; 其次, 国产化基础软件不太成熟, 国产操作系统、中间件等基础软件虽然均通过常规测试, 但是集成运行时仍然会出现系统死机、不稳定、长时间运行后系统性能下降等现象, 需要进行专项可靠性测试技术研究; 最后, Windows、Oracle 等非国产化基础软件面向全球用户使用, 其可靠性由公司内部测试与直接用户测试共同保证, 但国产基础软件正处于研制与推广中, 缺少用户使用角度的可靠性测试优势, 需要构造国产基础软件的使用剖面, 并生产可靠性测试用例进行可靠性测试.

① 收稿时间:2015-06-25;收到修改稿时间:2015-08-17

软件可靠性测试是指为了满足软件可靠性要求, 验证是否达到软件的可靠性要求, 评估软件的可靠性水平而对软件进行的测试, 常采用基于软件操作剖面对软件进行随机测试的可靠性测试方法[1]. 目前, 两种主要的软件可靠性测试方法包括 AT&T 贝尔实验室的 J.D Musa 在 1993 年提出的基于运行剖面的可靠性测试方法[2]以及 Harlan D.Mills[3]和 James A.Whittaker[4]提出的基于使用模型的统计测试方法, 如Markov链使用模型. 但是, 现有的可靠性测试方法无法直接应用于国产化基础软件可靠性测试中. 首先, 与常规软件不同, 用户是通过操作应用程序间接地域基础软件进行交互, 国产基础软件的运行剖面无法直接获得; 其次, 对于操作系统来说, 软件运行剖面中的运行或功能是操作系统的 API 序列, 而常规软件运行剖面中的运行或功能往往是用户的一个界面操作或软件的一个功能; 最后, Musa 剖面方法主要针对功能和运行简单, 输入序列独立, 不相关的批处理软件, 而操作系统的各个运行之间的关联和约束比较复杂, 构建出的Musa剖面与操作系统的真实使用情况相差甚远, Markov 链模型的优点是可以较好地表示功能或状态彼此不独立的软件运行情况, 但是该模型考虑更多的是软件的各种状态及状态之间的转换, 对整个操作系统的状态和状态之间的转换复杂, 使用Markov链使用模型进行分析时会导致模型的复杂度提高.

综上所述, 本文针对国产操作系统特性及可靠性特殊需求, 结合 Musa 剖面和 Markov 链模型来构建操作系统的运行剖面, 提出一种带标记的Markov链运行剖面建模技术, 然后按照运行剖面描述的软件运行方式测试软件所得的失效数据, 并选取可靠性模型进行可靠性评估.

1 国产器操作系统可靠性度量指标

目前, 国产操作系统难以得到必要的数据, 因此, 确定国产操作系统可靠性指标的基本原则为: “需要”与“可能”相结合, 经综合权衡后加以确定. 要考虑的方面包括: 用户要求、系统的重要程度、系统的可靠性要求、国内外软件可靠性的水平、相似软件的可靠性水平、软件管理、开发人员、软件开发技术水平以及开发工具的使用、进度要求、经费保障、指标要求的可验证性.

根据使用方对被测系统的可靠性要求, 在可靠性度量指标中选择需要验证的指标.

1) 操作系统可靠性运行首先应该保证在规定的条件下, 软件完成规定功能的概率, 所以选用可靠度作为一个可靠性度量指标;

2) 由于操作系统的失效频率对于系统的可靠性有着非常大的影响, 所以选择失效率/失效强度作为一个可靠性度量指标;

3) 操作系统的可靠性、可用性及可维护性能力属于重点关注的对比维度 RAS, 具体包括硬件故障检测和恢复能力、持续服务能力、系统崩溃恢复和分析能力以及 HA 集群能力 4 个方面. 针对 RAS 确定可靠性度量指标, 选择平均故障前时间(MTTF)和平均故障间隔时间(MTBF)作为可靠性度量指标.

综合以上情况, 根据国产操作系统的特点及使用要求, 确定国产操作系统可靠性指标体系, 如表 1 所示.

表 1 国产操作系统可靠性度量指标

一般的可靠性指标体系	国产操作系统可靠性度量指标
可靠度、失效率/失效强度、初期失效率、偶然失效率、平均失效前时间(MTTF)、平均失效间隔时间(MTBF)、平均失效修复时间(MTTR)、平均不工作时间(MTBD)、成功率、任务成功率(MCSP)、由平均失效前时间派生的参数、平均致命性失效前时间	可靠度、失效率/失效强度、平均失效前时间(MTTF)、平均失效间隔时间(MTBF)

2 国产操作系统可剖面建模技术

GJB7706-2012[5]中对军用嵌入式操作系统的可靠性测试内容作出简要的描述: “按照软件运行剖面(对软件实际使用情况的统计规律的描述)生成测试用例, 执行软件进行测试, 收集数据(包括输入数据、输出数据), 进行失效分析. ”

运行剖面是软件可靠测试的重要步骤, 也是生成可靠性测试用例进行可靠性测试的基础, 较为常用的的剖面建模型主要包括 Musa 模型和 Markov 链模型.

2.1 基于 Musa 的剖面构建方法

Musa 的软件划分原则简单易实施, 只要按照步骤逐步实行就可以得出软件的比较准确的运行剖面. 构建运行剖面分为五个步骤, 依次构建客户剖面、用户剖面、系统模式剖面、功能剖面以及运行剖面, 最后根据运行剖面生成测试用例[6].

具体步骤如下:

1) 根据元素定义划分剖面元素;

2) 估算每一个元素的概率值;

3) 合并同一层中相同的剖面, 将其概率值相加.

在构建运行剖面的过程中, 需要了解用户是如何使用该软件的. 要充分了解软件的各种功能以及这些功能在系统运行时发生的概率. 定义使用概率的最佳方法是使用实际的用户数据, 如来自原型系统、前一版本的使用数据; 其次是由该软件应用领域的用户和专家提供的预期使用数据.

2.2 基于带标记的 Markov 链的运行剖面建模技术

基于 Musa 剖面模型的构建方法自顶向下的将系统划分为五层剖面, 最终由运行剖面产生测试用例, 能方便有条理地得到用户对软件每个运行的使用情况. 但是, 这种方法主要针对功能和运行简单, 输入序列独立、不相关的批处理软件. 然而由于并行性和共享性等特点, 操作系统各个运行之间的关联和约束比较复杂, 构建出的 Musa 运行剖面与操作系统真实的运行情况会相差甚远. Markov 链模型的优点是可以较好地表示功能或状态彼此不独立、相互关联的软件的运行情况, 但该方法更多考虑的是软件的各种状态以及状态之间的转换, 对整个操作系统的各种状态都使用 Markov 链模型分析时, 其工作量往往很大, 且模型中较易出现状态和迁移的冗余. 本文将结合这两种模型设计出一种带标记的 Markov 链运行剖面建模技术来构建操作系统的运行剖面, 具体建模过程如图 1 所示.

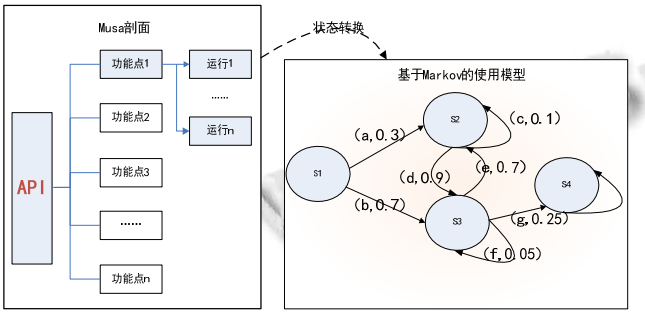

图 1 结合 Musa 剖面和 Markov 模型的运行剖面建模过程

首先从功能剖面开始对操作系统的基本功能点进行划分, 然后对基本功能点进行分解得到所有的系统运行, 再针对功能或运行相互关联的部分, 辅助采用基于Markov链模型的方法划分剖面, 以便能全面地描述操作系统各个运行之间的互连和转换关系.

1)运行的模型表示

本文用运行图来描述特定功能点分解后的运行关系, 定义为:

$$TF=\{P_1,P_2,...P_n\}$$

其中, $P_1,P_2,...,P_n$ 表示构成运行的各个状态, P_i 的下一个状态为 P_{i+1}, P_i 的上一个状态为 P_{i-1}, 这些状态表示一个任务从开始到结束的过程, 即 $P_1\rightarrow P_2\rightarrow ...\rightarrow P_n$. 当运行图中的某个状态可以有几种不同的路径到达下一个状态时, 仅用运行图就不能准确表达该运行了, 此时, 就要用到状态细化图.

状态细化图用来描述运行图中状态的内部细节, 定义为一个三元组:

$$DTF=(start,sequence,end)$$

其中, $sequence=\{B_i \mid B_i=TF_i\}, i=1,2,...,n$,*start* 为细化图的公共开始节点, end 为细化图的公共终止节点. 只要划分的足够细, 被测软件中的所有运行都可以由运行图和状态细化图准确地表示出来.

2)将运行图、状态细化图表示的运行剖面转化为 Markov 链表示

将运行图和状态细化图描述的运行剖面转化成 Markov 链描述主要是基于以下考虑:

首先, Markov 链的特点是下一个状态只和当前状态有关, 而与历史状态无关, 则若上一状态正确, 在正确的输入下当前状态一定正确, 否则, 软件存在缺陷. 这对于定位软件测试中的错误是十分方便的, 通过Markov链中状态转移概率, 还能直观地认识到软件中各个运行的使用频率, 给出一个定量的描述.

其次, 当某个节点内部有需要细化的分支时, Markov 链综合内部分支, 给出一个整体的表述, 这对于产生测试用例非常方便.

3 国产操作系统运行剖面构建及可靠性测试用例生成

3.1 操作系统运行剖面构建

操作系统的主要功能点共有八个, 包括: 任务管理、任务同步与通信、时钟/定时器管理、中断/异常管理、内存管理、文件系统、设备管理和网络通信[7]. 在实际的应用研究中, 可以根据被测系统的需求文档和软件来确定主要功能点. 通过运行应用负载, 对这八个功能进行统计、排序并设置权值.

在功能剖面中, 功能不是一个可以实际运行的实

体, 把功能剖面继续分解成的运行剖面才是可以实际运行的单元. 我们可以把运行理解为功能剖面下的一个具体的小功能, 功能剖面是一个集合, 这个集合中的元素是一个个运行. 因为功能剖面下的具体运行在软件的需求分析阶段就已确定, 我们只需要按照功能剖面的划分, 即可将每一个功能剖面都分解得到具体的运行. 以任务管理为例进行主要运行的划分, 如表 2 所示.

可以通过公式 $P_{ci}=\frac{c_i}{\sum_{j=1}^{n}c_j}(i=1,...,n)$ 计算出任务管理功能下每一个运行对应的概率值. 这样, 每一运行最终的概率值 $P_{oi}=P_f*P_{ci}$, 其中, P_t 为任务管理功能对应的概率.

表 2　任务管理功能下的主要运行划分

功能模块	主要运行	出现次数
任务管理功能	创建任务	C1
	使任务就绪	C2
	删除任务	C3
	唤醒任务	C4
	挂起任务	C5
	恢复任务	C6
	获得任务 ID	C7
	分配任务堆栈	C8
	设置任务优先级	C9
	任务睡眠一段时间	C10
	任务睡眠到指定时间	C11
	设置笔记本	C12
	获得笔记本	C13
	记录任务变量	C14
	获得任务变量	C15
	删除任务变量	C16

在划分运行剖面之后需要对运行的转换关系进行描述. 以创建任务的运行为例, 由于阻塞的原因不同, 阻塞态应该分为 6 种不同的状态, 这 6 种状态中, 挂起状态又是一个特殊的状态, 因为从创建状态转换的挂起状态、从就绪状态转换的挂起状态和从运行状态转换的挂起状态是不同的, 要视为 3 种不同状态. 这样从阻塞态就分出 8 种不同的状态. 如图 2 所示. 任务处于创建态、就绪态和运行态时都可以被挂起而引起阻塞, 但系统对这三种阻塞态的处理是不同的, 所以可看作三种状态. 可将这三种状态称为: 由创建态挂起的阻塞态、由就绪态挂起的阻塞态和由运行挂起的阻塞态. 其中“由运行挂起的阻塞态”和“从运行态转换的阻塞态”一同作为阻塞态处理, 而前两者分别作为两种挂起状态处理.

从就绪态到运行态存在三种情况, 分别是: 任务抢占其它低优先级任务而运行; 任务通过非抢占方式从就绪态转换到运行态; 当采用时间片轮转时, 达到任务的时间片时, 任务从就绪态转换到运行态[8]. 操作系统对这三种情况而来的运行态的处理也是不同的. 所以, 我们也把通过不同就绪态转换的运行态区分开来, 有三种状态, 不过这三种状态在转换到其它状态时的情形是相同的, 所以在处理的时候, 把它们看作一个整体. 从运行态到阻塞态存在六种情况: 调用 XX_task_suspend 导致任务阻塞; 调用 XX_task_wake_afte 导致任务阻塞; 任务试图获取一个信号量, 但请求没有被满足, 任务被阻塞; 任务试图从一个区(region)中获取一个段(segment), 但请求未被满足, 任务被阻塞; 任务试图从消息队列中获取消息, 但是队列中没有消息, 任务因为等待消息而 被阻塞; 任务试图获取一个事件(event), 但该事件没有发生, 任务因为等待事件发生而被阻塞.

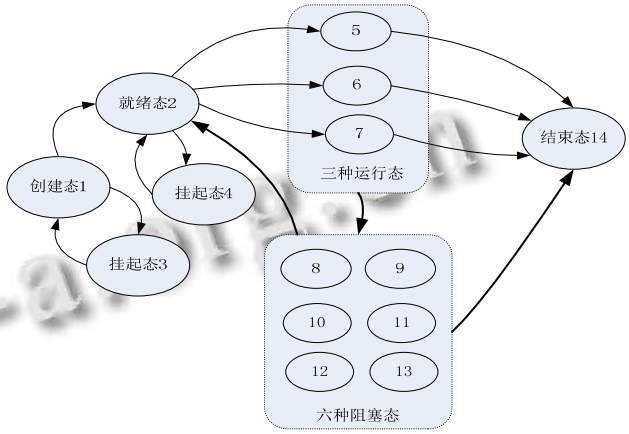

图 2　操作系统内存管理状态转换图

3.2 剖面的形式化表示

操作系统的剖面划分需要合适的形式语言描述, 以便计算机读取, 并在计算机内以适宜的数据结构存储. 本文使用类 XML 的一种形式语言, 实现对操作系统运行剖面的描述. 为每个运行点分配大于 0 的唯一序号值, 通过界定能到达的下个运行点的序号来定义运行点之间的指向和路径. 定义下个运行点序号值为 -1 时, 表示该运行点无指向下个运行点的路径, 即为运行的终点. 运行点有众多下个运行点时, 逐一分配

到下个运行点的概率. 形式语言描述如下:

```
- <剖面>
  <运行>
    <运行序号> 运行 1  </运行序号>
    <初始化及约束>  初始化及约束 </初始化及约束>
    <运行名> 运行名 </运行名>
    <运行简要描述> 运行简要描述 </运行简要描述>
    <预期结果> 预期结果 </预期结果>
      <输入变量>
        <变量名> 变量名 1 </变量名>
        ……
        <变量名> 变量名 n </变量名>
      </输入变量>
      <下个运行>
        <运行范围>
          <下个运行序号> 下个运行序号 i </下个运行序号>
          <运行概率> 运行 1 转移至运行 i 的概率 </运行概率>
        </运行范围>
        ……
        <运行范围>
          <下个运行序号> 下个运行序号 j </下个运行序号>
          <运行概率> 运行 1 转移至运行 j 的概率 </运行概率>
        </运行范围>
      </下个运行>
  <运行>
    <运行序号> 运行序号 n  </运行序号>
    ……
  </运行>
</剖面>
```

根据已构造的运行剖面, 列出所有的运行点、运行点能到达的下个运行点及其概率、运行点对应的变量值域范围和概率, 对每个运行点信息均按形式语言的格式描述. 对于安全关键运行点, 可酌情提高转移至该运行点的概率, 以增加被运行到的几率. 测试用例是根据运行剖面随机生成的. 根据随机测试的原则, 在运行剖面给定的输入变量的取值区间内任意抽取一个变量值, 将各个变量按顺序组合起来便生成了测试用例.

3.3 基于 Markov 剖面的测试用例自动生成技术

基于 Markov 模型构建的剖面非常适合生成测试用例, 这是由该模型的特点决定的. 该模型的各种状态之间的转换有一个概率值与之对应, 因此从初始状态开始, 生成一个[0, 1]区间内的随机数, 根据随机数确定当前状态转换的下一个状态, 直到下一个状态为终止状态, 便可获得一个测试用例. 根据 Markov 模型的特点, 随机选择路径.

具体方法为:

第一步: 根据状态图构建一个转移概率矩阵 $P=[P_{ij}]_{n\times n}$, 假设该模式下有 n 个状态, 则应构建一个 n*n 矩阵, 并将状态的转移概率赋值给对应的矩阵元素, 其中, P_{ij} 表示状态 i 到状态 j 之间的转移概率.

第二步: 将状态 $i(i=1,2,...,n)$所有出边的转移概率值和[0, 1]区间的某个区间段联系起来, 区间的长度等于状态 i 某个出边 j 发送的概率 P_{ij}.

第三步: 从初始状态出发, 产生一个[0, 1]区间的随机数, 根据该随机数所属的区间, 选择下一个转移状态.

第四步: 重复第三步, 直到下一个状态为终止状态.

在随机选择路径之后, 以同 Musa 剖面模型一样的方法, 从这些路径中提取影响这些路径的输入变量. 对这些变量按取值区间进行等价类的划分. 之后随机选择等价类, 并以均匀概率从选中的等价类中选择输入状态, 之后生成测试用例, 具体的流程如图 3 所示.

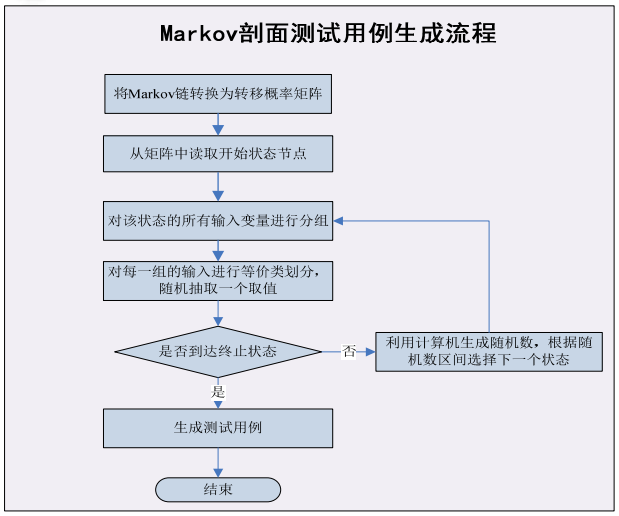

图 3 Markov 模型部分测试用例生成流程

根据上述方法产生测试用例, 执行可靠性测试, 测试完全根据各个运行所发生的概率以及运行的权重来进行的, 在测试中, 优先测试那些最重要或最频繁使用的功能和运行, 释放和缓解最高级别的风险, 有助于尽早发现那些对可靠性有最大影响的失效.

4 可靠性分析及应用验证

本文选取基于龙芯 2F 的 DeltaCORE 3.2.4[9,10]作为可靠性测评的验证对象. 根据可靠性测试的步骤, 在完成剖面的构建以及测试用例的生成后, 并用这些测试用例驱动软件测试之后, 收集测试中的失效数据, 再根据软件的失效数据利用可靠性模型对可靠性进行评估.

在 DeltaCORE 3.2.4 的第三方测试阶段和可靠性测试阶段共收集失效数据有 17 组, 具体采集的失效数据记录如表 3 所示. 其中, x 是发生失效的累积时间(单位: 小时), y 是累积发生的失效数, t 是发生失效的间隔时间(单位: 小时).

表 3 失效数据及时间数据

累积失效数y	时间间隔 t	累积时间 x	累积失效数y	时间间隔 t	累积时间x
1	1	1	10	8	105
2	2	3	11	32	137
3	1	4	12	20	157
4	5	9	13	14	171
5	6	15	14	22	193
6	24	39	15	76	269
7	10	49	16	50	319
8	15	64	17	92	411
9	33	97			

将失效数据及各个运行的测试用例数及失效数带入测试充分性的计算公式中有:

$$FI=\frac{3\times 0.0307}{8404\times 0.0307}+\frac{2\times 0.029}{9862\times 0.029}+\ldots+0=0.059$$

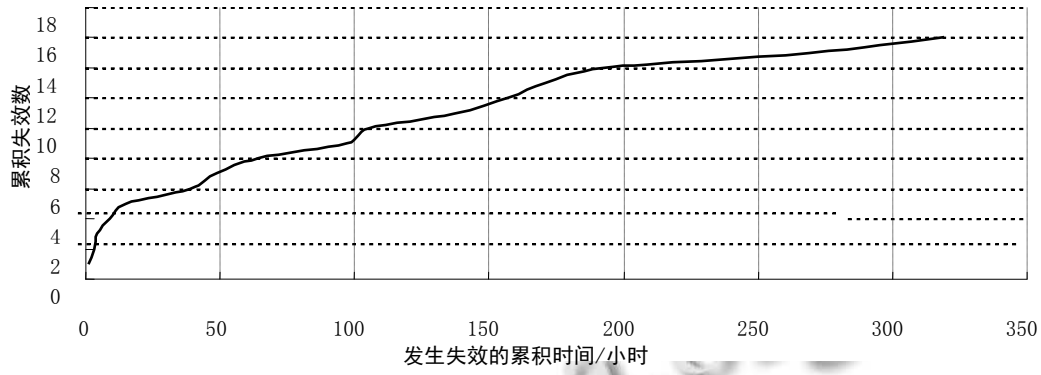

图 4 失效数据散点图

即可靠性测试的充分性已经超过给定的指标值 0.05, 停止可靠性测试, 并根据可靠性测试得到的失效数据对 DeltaCORE 3.2.4 的可靠性进行分析和评价.

取前 16 组原始数据进行拟合, 第 17 组数据进行验证, 作出如图 4 所示的失效数据散点图. 从图中可以看出, 累积失效数的增长曲线呈现指数或者幂函数的趋势, 即开始增长缓慢, 然后快速增长, 最后趋于饱和.

从图 5 以看出, 失效数据的散点图接近指数函数或幂函数, 因此选择趋势较为接近的 G-O 模型和 Duane 模型进行拟合. 根据这两种模型的拟合效果, 本文最终选择 G-O 模型对原数据进行拟合, 并对被测系统进行可靠性分析和评价. 图 5 给出了原始失效数据曲线和拟合的 G-O 模型曲线. 从图中看出, 所建立的可靠性度量模型与原始数据拟合较好, 为了给出拟合程度的量化数据, 还需要进一步计算回归曲线的相关系数, 即模型的拟合优度值.

在选择 G-O 模型作为原始数据的拟合模型后, 可以根据该模型对基于龙芯 2F 的 DeltaCORE 3.2.4 的可靠性进行预测和分析, 以辅助评价或控制软件的质量. 根据上文中构件的操作系统可靠性评价指标体系, 可以计算出 DeltaCORE3.2.4 的任务管理功能可靠度达到 88.78%.

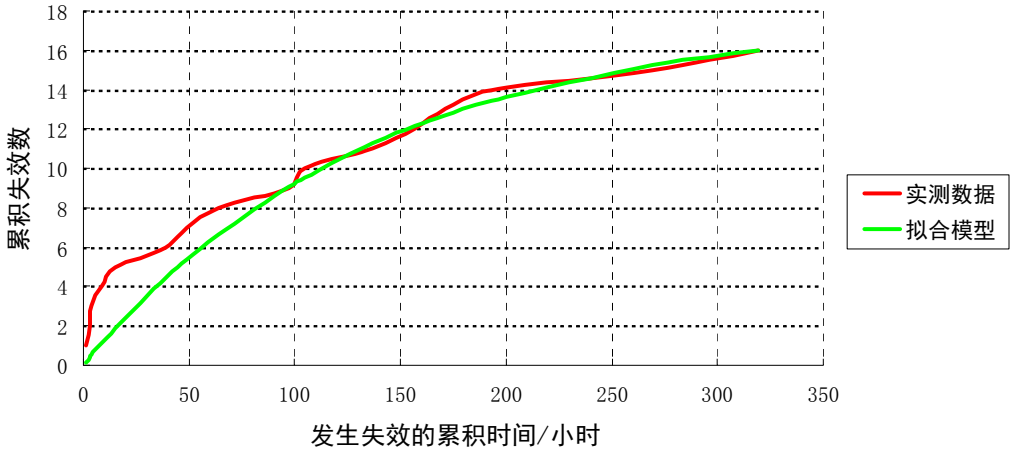

图 5 原始失效数据与 GO 模型的拟合曲线

5 结论

本文针对国产基础软件特性及可靠性特殊需求, 提出一种国产基础软件可靠性分析的方法; 从操作系统的不同运行剖面层次进行剖面建模技术研究, 并最终提出一种带标记Markov链的运行剖面建模技术. 并选取基于龙芯2F的DeltaOS作为验证对象进行应用验证研究, 理论联系实际, 使成果得到了及时的应用和很好的验证.

参考文献

1 李学仁.军用软件质量管理学.北京:国防工业出版社,2012.

2 Musa JD. Operational profiles in software-reliability engineering. IEEE Software, 1993, 10(2): 14–32.

3 Mills HD, Dyer M, Linger RC. Cleanroom software engineering. IEEE Software, 1987, 9: 19–24.

4 Whittaker JA, Poore JH. Statistical testing for cleanroom software engineering. Hawaii International Conference on System Sciences. 1992.

5 GJB 7706–2012.军用嵌入式操作系统测评要求.国家军用标准局,2013.

6 黄锡滋.软件可靠性、安全性与质量保证.北京:电子工业出版社,2002.

7 兰雨晴,赵同,高静,等.基础软件平台质量评估.软件学报, 2009,20(3):567–581.

8 刘玮,兰雨晴.国产基础软件兼容性测试方法研究. 2006 IBM 用户大会论文集,计算机系统应用,2006,12(增刊): 182–185.

9 DeltaCORE3.2.4 用户手册.

10 DeltaCORE3.2.4 参考手册.