基于监控的软件动态可靠性①

陈 镭, 唐 晶

(南京审计学院 图书馆, 南京 211815)

摘 要: 在开放环境下, 软件系统的行为更加的多样化和复杂化, 而传统利用体系结构评估系统可靠性的方式大都在部署前实施, 均基于对运行环境、用户需求的不变性假设. 为此, 本文通过监控发现软件行为与需求、测试不一致的地方, 配合软件可靠性模型, 动态评估系统在运行期间的可靠性. 文中提出风险因子的概念和一种资源分配的方法, 进而保障系统可靠运行, 实验数据验证了本文提出方法的有效性.

关键词: 软件模块; 可靠性模型; 软件监控; 风险因子; 资源分配

Dynamic Reliability for Software System Based on Monitoring

CHEN Lei, TANG Jing

(Dept. of Library, Nanjing Audit University, Nanjing 211815, China)

Abstract: In open environment, the system behaviors are diversified and complicated, but traditional static models are mostly implemented before the deployment. They are based on certain invariance assumptions of the operating environment and user needs. We study on the monitoring of software system to find the differences of software reliability between testing phase and running phase, combining static module-based software reliability model to evaluate the system reliability during the running phase. Gives the risk factors that decrease module reliability and proposes an effective method to reallocate the resources. Through the experimental data, verifying the effectiveness of the proposed method.

Keywords: software module; reliability model; software monitor; risk factor; resource allocation

1 引 言

软件系统的规模和结构正变得庞大和复杂, 系统的开发大量借助于软件模块的重用. 因此, 出现了大量的基于模块的软件可靠性模型, 这些模型可分为三类: 基于状态的模型, 基于路径的模型和基于模块概率迁移图的模型. 基于状态的模型采用分析模块状态转移的方式, 这类模型假设模块之间的控制转移具有马尔科夫属性, 软件体系结构建立模型时使用离散时间的马尔科夫链、连续时间的马尔科夫链或半马尔科夫过程, 基于状态的模型代表性的有: Littlewood 模型[1], Cheung 模型[2]. 基于路径的模型, 除了建立模型时体系结构与失效结合的方法与基于状态的模型不同外, 其他通用步骤和基本思想是相似的. 基于路径的模型代表性的有: Shooman 模型[3], Krishnamurthy 模型[4]. 基于模块迁移概论图的模型[5], 以模块可靠性为基础构建函数抽象模型和模块概率迁移图.

但是, 这些可靠性模型均建立在用户需求、运行环境等的一些不变性假设基础之上, 而系统设计和测试技术很难覆盖到所有的应用场景[6], 所以我们认为, 开放运行环境中软件系统的可靠性需要“把握”和“调整”, 一个可行的途径是“监控与演化”[7]. 通过监控及时发现软件运行过程与需求、测试不一致的地方, 进而进入人为或自动的生命调整周期, 确保系统在宏观和微观层次上满足要求, 有效防止软件衰老甚至失效, 保障系统平稳运行.

2 动态的评估系统可靠性

2.1 评估系统可靠性的方法

① 收稿时间:2013-10-15;收到修改稿时间:2013-11-18

通常把基于状态的模型和基于路径的模型概括为一通用模型:

$$R_s = \prod_{i=1}^{n} R_i^{V_i} \tag{1}$$

从公式(1)可看出, 系统可靠性只与模块的可靠性和模块之间的转移概率有关. 在开放的运行环境中, 存在很多不确定因素, 导致模块自身可靠性 R_i 和模块之间的转移概率 V_i 都很难与需求测试保持一致性, 采用监控的方法能弥补静态模型的局限.

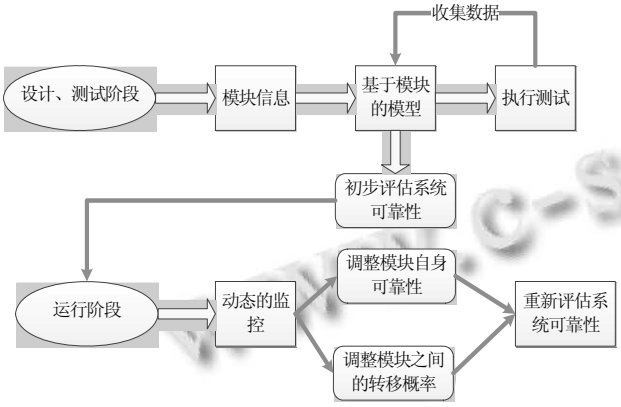

图 1　动态评估系统可靠性流程图

图 1 描绘了动态评估系统可靠性的流程. 设计阶段, 根据系统设计图, UML 图, 确定模块的粒度; 测试阶段, 通过黑盒, 白盒及大量的测试用列, 获得模块自身可靠性和模块之间的转移概率, 作为可靠性模型的输入参数, 测试结束后计算出系统可靠性; 运行阶段, 通过监控获取系统运行剖面与测试剖面的差异, 定期的更新模块自身的可靠性和模块之间的转移概率, 进而构建离散时间的马尔科夫链(DTMC),结合公式(1)动态评估系统运行期间的可靠性.

2.2 风险因子[8]

模型的精确性取决于模型假设和现实情况的差距, 公式(1)的建立存在以下假设:

① 各模块之间的转移概率符合马尔科夫过程.

② 模块的失效相互独立, 一个模块的失效不影响到其他模块.

③ 测试用例覆盖了所有可能的应用场景, 即测试剖面与运行剖面相同.

正是因为第 3 条假设的存在, 所以运行期间系统可靠性存在一种合理的(测试剖面和运行剖面不会存在过大差异[9])下降趋势. 定义 $Rist_i$(风险因子)为模块 i 可靠性降低的风险程度, $Rist_i$ 越大, 模块 i 可靠性降低的可能性越大. $Rist_i$ 的值取决于: 1)监控周期 T 内发现的测试剖面与运行剖面存在差异的操作数. 2)监控对象的个数, 包含 web 响应时间, cpu 时间, 数据库响应时间, 内存利用率等, 监控的对象越多, 结果越精确. 3)软件测试过程的可信度, 测试工具是否先进, 测试方法是否有效. 利用监控工具, 前两个因素较易获得. 在监控周期 T 内测试剖面与运行剖面的差异数越多, 监控的对象越多, $Rist_i$ 越大. 第 3 因素, 和软件测试过程的规范度有关, 其值由测试人员指定, 取值范围(0, 1).

风险因子 $Rist_i$ 为:

$$Rist_i = \frac{DifferenctActions}{AllActions} \times \frac{DifferenctPoints}{AllPoints} \times W \tag{2}$$

在监控周期 T 内, Allactions 代表所有操作数, DifferenctActions 为运行剖面和测试剖面存在差异的操作数目, 超出给定的范围就认为存在差异; AllPoints 表示监控对象的总数目, DifferenctPoints 为存在差异的对象数目; W 则是对测试过程的信任程度, 取决于测试工具是否先进, 测试方法是否有效.

N 个监控周期后, 模块 i 的可靠性为:

$$R_{ONLINE_i}^{n} = R_{ONLINE_i}^{n-1} - R_{ONLINE_i}^{n-1} \times Rist_i \tag{3}$$

运行期软件可靠性通用公式为:

$$R_{ONLINE_S} = \prod_{i=1}^{n} R_{ONLINE_i}^{V_{ONLINE_i}} \tag{4}$$

其中 $V_{ONLINEi}$ 为模块 i 真实访问频率, 通过对模块的行为监控能准确计算出在间隔 T 内的 $V_{ONLINEi}$.

3　调整资源分配

3.1 敏感性分析

基于模块的可靠性建模方式下, 不同模块对系统的影响存在差异, 某些模块可靠性的变化对系统可靠性影响较大, 所以想要提高系统可靠性, 应着重关注那些关键模块.

假设下式成立[10]:

$$\left|\frac{\delta R_S}{\delta R_i}\right| \geq \left|\frac{\delta R_S}{\delta R_j}\right| \quad , \text{其中 } j = 1,2,\ldots,n.$$

式中 $\&R_S$ 为系统可靠性的变化, $\&R_i$ 为模块 i 可靠性的变化, $\&R_j$ 为模块 j 可靠性的变化.

定义 $S_{P,Ri}$ 为某一模块可靠性 R_i 增减 p%后, 系统可

靠性 R_S 对应的增减量. 则 $S_{P,Ri}$ 为:

$$\delta R_S = S_{P,R_i} = \frac{\left|R_s(R_1,R_2,..,R_i+p\times R_i,..,R_n)-R_s(R_1,R_2,..,R_i,..,R_n)\right|}{R_s(R_1,R_2,..,R_i,..,R_n)}\times 100\%$$

假设的不等式另一种表示形式为:

$$\left|\frac{S_{P,R_i}}{P\%}\right| \geq \left|\frac{S_{P,R_j}}{P\%}\right|$$ ，其中 j = 1,2,…,n.

根据以上两个式子, 可得到不等式

$(1+p)^{Vi}>=(1+p)^{Vj}$,当 p>0 和不等式

$(1+p)^{Vi}<=(1+p)^{Vj}$,当 p<0.

以上不等式为一幂指函数 $y=x^n$,(x>0), 当 p>0 时, 1+p>1,即 x>1, 幂指函数 $y=x^n$,(x>0)为增函数, V_i 越大, 函数值越大; 当 p<0 时, 0<1+p<1,即 0<x<1, 幂指函数 $y=x^n$,(x>0)为减函数, V_i 越大, 函数值越小. 至此可得结论, 某一特定监控周期 T 内, 模块被访问的频率 V_i 越大, 模块可靠性 R_i 在变化相同比例 p%的情况下, 对系统可靠性影响越大, 称此模块敏感.

3.2 资源分配算法

实际应用中, 资源与可靠性的函数模型关系比较复杂, 一种现有的资源—可靠性模型为:

$$r_i = p_i R_i^{q_i}$$

其中 p_i 和 q_i 为与模块有关的常数. 定义 f_i 为模块在当前状况下提高可靠性的难易度, f_i 的值应该为资源函数 r_i 函数曲线在值 R_i 处切线的斜率值:

$$f_i = r_i' = p_i * q_i * R_i^{(q_i-1)}$$

可以看出, f_i 的值只与模块可靠性 R_i 有关. f_i 的取值范围(0, 1), 显然花费同样资源, f_i 值越大, 提升模块可靠性越困难.

综上分析, 基于模块的软件系统中, 要提高系统可靠性, 模块访问频率 V_i 和模块可靠性提升难易度 f_i 这两个因素起到决定性的作用.

算法的具体步骤如下:

① 按模块访问频率 V_i 值由低到高对所有模块进行排序.

② 使用可靠性通用公式 1 计算系统可靠性. 如评估值低于警戒值, 则转到步骤 3; 如评估值高于警戒值, 则不需调整资源分配, 进入下一个监控周期.

③ For (i=0; i<n; i++),综合考虑 f_i 的情况下, 从模块 i 移除单位资源, 计算移除单位资源之后的可靠性 R_i'.

④ For (j=n; j>0; j--),综合考虑 f_j 的情况下, 增加单位资源给模块 j, 计算增加单位资源之后的可靠性 R_j'.

⑤ 循环结束后找到拥有最大 $(R_i')^{V_i} * (R_j')^{V_j}$ 值的模块 i 和模块 j, 从模块 i 移除单位资源分配到模块 j.

⑥ 重复执行步骤 2.

4 实验分析

Glassbox 是一款开源, 基于 AOP 技术的 J2EE 监控软件. 对一些使用第三方模块的复杂、多线程、分布式 Java 系统, 如请求数目, 执行时间, 缓慢、失效数之类等, Glassbox 也能深入的监控. 另外 Glassbox 带来的开销相对较小, 对性能的影响在可接受范围内.

监控的目标系统是我们自行开发的一个网上售书系统, Java 语言编写, 代码规模超过 1 万行, 系统共有 10 个不同功能的模块组成. 使用 Tomcat 5.5 作为 Web 服务器, 数据库 Oracle 11g, 采用 Jdbc 方式连接.

按照本文中提出的方法, 首先计算测试阶段系统的可靠性. 我们编写了 100 组测试用例, 每组测试用例平均执行时间为 300s, 分配到三台处在在同一局域网内的主机访问系统, 测试阶段系统可靠性 Rs=0.9688. 本实验中, 我们设置 R_{dec}(用户能够接受系统可靠性降低的最大限度)为 2%, 这样运行期间用户能够接受的系统可靠性最低值为 $R_{min}=Rs-R_{dec}=0.9688*(1-2\%)=0.95000$. 我们共进行了 20 次独立实验, 监控周期选择 T=3600 秒, 每次实验监控 8 个周期, 可得每次实验监控的总时长为 3600*8=28800s.

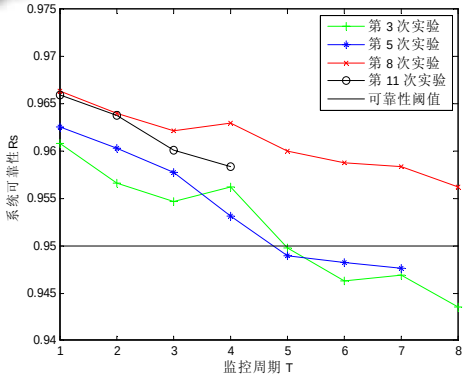

图 2 系统可靠性变化的 4 种结果

20 组实验得到的结果可分为 4 种情况:

正常运行, 没有触发警报, 也没有导致系统失效,

如第 8 次实验; 触发警报, 但是没有导致系统失效, 如第 3 次实验; 触发警报, 同时导致系统失效, 如第 5 次实验; 没有触发警报, 但导致系统失效, 如第 11 次实验, 这四种情况的比例为 7: 3: 8: 2. 图 2 还可以看出, 虽然第 5 次实验引起系统失效, 但从系统发出警报到最终失效还经历了 3 个监控周期 T, 周期 5, 6, 7, 和第 5 次实验同一类的另外几组实验也大都存在这样的现象. 这给了我们启示: 系统在运行期间发出报警后, 根据相关规则自动或人工的进行调整和配置, 更换热部件, 调整冗余资源, 甚至暂停运营等补救措施, 是完全可以避免系统失效, 保障服务可靠运行的.

表 1 资源分配过程中模块可靠性的变化

	R_2	R_8
0	0.99770	0.99710
1	0.99775	0.99704
2	0.99780	0.99699
3	0.99785	0.99695
4	0.99790	0.99690
5	0.99795	0.99685
6	0.99800	0.99680

我们以第 3 次实验中第 5 个监控周期末的数据为基础, 仿真实现了 3.2 节中提出的资源分配算法. 从表 1 可以看出, 第 1 次重新调整资源后, 从模块 8 移除单位资源分配给模块 2, 模块 8 的可靠性由 0.99710 下降到 0.99704; 另一方面, 模块 2 得到了新资源, 其可靠性由 0.99770 上升到 0.99775, 套用运行期可靠性通用公式, 计算出此时系统可靠性为 0.94987, 已经高于未调整资源分配之前的系统可靠性 0.94983, 但还未达到用户能够接受的最低要求 0.95000, 所以继续调整资源, 最终经过 6 次调整过程后, 系统可靠性达到了 0.95180, 满足了用户要求.

5 结 语

本文对运行期间软件系统的可靠性问题开展研究, 结合静态基于模块的软件可靠性模型和动态行为监控的方法, 定期评估在线软件系统可靠性. 通过对监控数据进行分析, 指出运行期间系统可靠性呈下降的趋势, 并给出模块可靠性降低的风险因子. 当系统可靠性降低到不能满足用户要求的时候, 在同时考虑模块敏感性和可靠性—资源关系的基础上, 提出了一种资源分配的方法, 指导分配资源, 保障系统可靠的运行.

参考文献

1 Littlewood B. Software reliability model for modular program structure. IEEE Trans. on Reliability, 1979, 28(3), 241–246.

2 Cheung RC. A user-oriented software reliability model. IEEE Trans. on Software Engineering,1980, 6(2): 118–125.

3 Shooman M. Structural models for software reliability prediction. Proc. 2nd Int'l Conference on Software Engineering. 1976. 268–280.

4 Krishnamurthy S, Mathur AP. On the estimation of reliability of a software system using reliabilities of its components. Proc. 8th Int'l Symp. Software Reliability Engineering. 1997. 146–155.

5 Yacoub S, Cukic B, Ammar H. Scenario-based reliability analysis of component-based software. Proc. 10th Int'l Symp. Software Reliability Engineering. 1999. 22–31.

6 陆文,徐锋,吕建.一种开放环境下的软件可靠性评估方法. 计算机学报,2010,33(3):452–462 .

7 陈镭.在线软件系统的动态可靠性研究[硕士学位论文].重庆:重庆大学,2012.

8 Pietrantuono R, Russo S, Trivedi KS. Online monitoring of software system reliability. Valencia, Spain, Eighth European Dependable Computing Conference. 2010. 28–30.

9 赵靖.考虑测试与运行差别的 NHPP 类软件可靠性增长模型研究[博士学位论文].哈尔滨:哈尔滨工业大学,2006.

10 Lo JH, Huang CY. Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. The Journal of Systems and Software, 2005,76: 3–13.