基于检查点和 Rejuvenation 的软件可靠性建模分析

范新媛，徐国治，应忍冬
(上海交通大学电子工程系，上海 200030)

摘 要：处于运行阶段的软件往往存在老化现象，即性能的逐渐下降或失效。为了避免软件老化造成的影响，人们提出了软件 rejuvenation 技术。这是一种预防性的软件容错策略，能有效的提高软件系统的可靠性和可用性。但由于 rejuvenation 操作过程的复杂性，它的成本也是不可忽略的。相对于 rejuvenation 来说，检查点是一种轻量级的软件容错策略，其成本远小于 rejuvenation 的成本。本文旨在结合 rejuvenation 和检查点技术，在 rejuvenation 周期间插入适当数目的检查点，以降低系统的宕机成本。文中给出了系统的 petri 网模型，并结合实例进行了分析。

关键词：软件可靠性；软件老化；软件 Rejuvenation；检查点
文章编号：1004-731X (2003) 11-1543-04　　**中图分类号：**TP206.3　　**文献标识码：**A

Analysis of Software Reliability Modeling on Checkpointing and Rejuvenation

FAN Xin-yuan, XU Guo-zhi, YING Ren-dong
(Department of Electronics Eng., Shanghai Jiaotong Univ, Shanghai 200030, China)

Abstract: Software systems often experience an “aging” phenomenon, characterized by progressive performance degradation or a sudden hang/crash. To counteract this phenomenon, a preventive approach named “software rejuvenation” has been proposed. Despite its effectiveness to improve software reliability and availability, rejuvenation is a complex and expensive process. Relative to rejuvenation, checkpointing is a light-level software fault-tolerance policy, the cost of which is much less than that of rejuvenation. This paper presents a new idea to combine both these techniques to further reduce the total downtime cost of software systems. Software reliability models for three cases and their corresponding numerical results are also given.

Keywords: software reliability; software aging; software rejuvenation; checkpointing

引 言

目前，计算机系统已广泛应用于社会生活的各个领域，功能越来越强大，复杂程度也不断提高。与此同时，人们对各类计算机应用系统的可靠性要求也越来越高。计算机系统的可靠性问题包括硬件和软件两个方面，硬件系统的可靠性研究已有成熟的模型和分析方法，而软件的可靠性和可用性问题还未有一个很好的分析解决方案。因此软件可靠性的研究已经成为制约系统可靠性的一个重要问题。

软件可靠性工作贯穿整个软件生命周期，从开发、测试，到发布、维护，而由于商业目的，一个软件的发布时间希望越早越好，这就必然缩短测试和调试过程，而且随着软件重用的普及，一般来说，不可能去多次测试应用程序所基于的操作系统和中间层软件。这些因素必然造成投入运行的软件总是存在一些残留的缺陷，例如内存泄漏，未释放锁定的文件，文件描述符的泄漏等。在运行过程中，这些缺陷的累积会导致软件的性能逐渐下降，甚至会使整个系统失效。这种现象就称为软件老化。

为了抵消软件老化造成的影响，Yennun Huang 等提出了软件 rejuvenation 技术[1]。每隔一段时间，重新启动程序，使之进入一个正常的初始状态，从而避免软件老化引起的失效。Rejuvenation 的过程包括停止应用程序的运行，释放操作系统的资源，刷新操作系统资源列表，重新初始化应用程序中的数据结构等。

虽然 rejuvenation 不能消除软件的残留缺陷，但它是一种有计划的停机，成本可控（其成本包括 rejuvenation 期间损失的收益以及执行 rejuvenation 所需的成本），引起的损失远小于突发的系统失效而带来的损失，因此软件 Rejuvenation 策略对降低总的系统宕机成本非常有利。

检查点 (checkpointing) 是计算机系统中一种常用的软件容错技术[2,6]。在程序运行过程中，每隔一段时间把当前的运行状态存储到磁盘上，以便在发生故障后程序可以从该状态恢复运行（卷回），而不需要从最开始重新执行整个任务。检查点带来的时间开销主要包括把程序的状态写入磁盘所花费的时间和卷回操作的时间。当程序运行与检查点操作完全串行执行时，检查点操作会使系统处于不可用的状态。

相对于 rejuvenation 而言，检查点操作的过程简单，是一种轻量级的操作，因而它引起的宕机成本小于 rejuvenation 的成本。但检查点只能解决瞬态错误引起的系统宕机，不能改变软件老化的过程。而 Rejuvenation 虽然

收稿日期：2002-10-17　　**修回日期：**2003-01-21
作者简介：范新媛(1977-), **女**, **湖南攸县人**, **博士生**, **研究方向为软件可靠性**; **徐国治**(1941-), **男**, **上海人**, **教授**, **博导**, **研究方向为软件可靠性、嵌入式系统设计等**; **应忍冬**(1975-), **男**, **上海人**, **博士生**, **研究方向为软件可靠性。**

可以减少软件老化引起的突发性系统失效，但单独使用代价太大，不利于系统性能的提升。

本文旨在结合 rejuvenation 和检查点技术，通过在 rejuvenation 周期间插入适当数目的检查点，以降低系统的宕机成本。本文第二部分建立了存在老化的软件系统模型，第三部分给出了采用不同策略时的系统 petri 网模型，第四部分对仿真实验的结果进行了分析，第五部分对文章进行了总结。

1 存在老化的软件系统模型

软件老化过程是指在软件运行过程中，随着时间和/或工作负载的变化，软件的性能逐渐下降，甚至导致软件失效和系统宕机。软件老化的随机过程模型如图 1 所示。目前已经提出了几种方法来检测软件的老化过程[3-4]，例如通过测量已用缓存和剩余内存空间等性能指标随时间和/或负载的变化规律来大体上反映这一过程，但总体来说，并没有一个精确的数学模型来描述软件老化的过程。关于 rejuvenation 的大部分文献[1,5]都用两阶段的过程来描述软件老化引起的系统失效，采用连续时间马尔可夫链（CTMC）对软件老化过程建模。

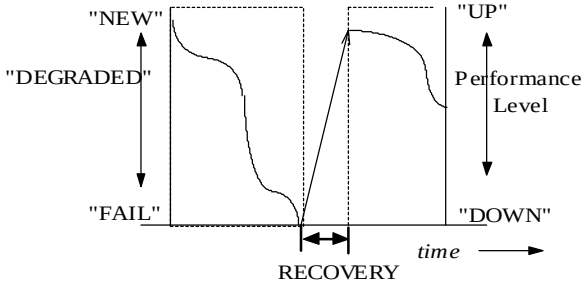


图 1　软件老化的随机过程模型

在本文中我们对[1]中的两阶段模型加以改进，用一个阶梯函数来近似软件失效率 $l(t)$ 随时间变化的过程，如图 2。假设在软件生命周期初期（0，t_1），软件的失效率为 l_0；在区间（t_1，t_2），处于运行阶段的软件的失效率为 l_1；在区间（t_2，t_3），软件的失效率为 l_2，…，并且 $l_0 \le l_1 \le l_2 \cdots$。基于这一假设，图 3 给出了软件老化过程的随机 petri 网模型描述。（随机 petri 网（SPN）中的白色矩形表示随机变迁，黑色矩形表示定时变迁，线段表示瞬时变迁。）

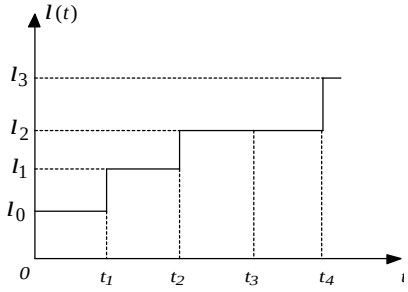


图 2　软件失效率 l 和时间的关系

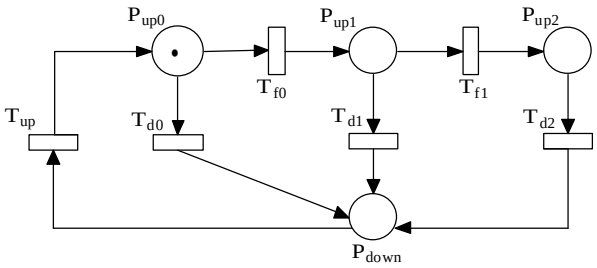


图 3　存在老化的软件系统模型

图 3 中，位置 $P_{up0} \sim P_{up2}$ 均表示系统处于工作状态，位置 P_{down} 表示系统处于失效状态。初始时刻系统处于 P_{up0} 状态，当变迁 T_{f0} 触发，系统进入 P_{up1} 状态，表示由于软件老化系统性能下降，但仍能提供服务。当变迁 T_{f1} 触发，系统进入 P_{up2} 状态，性能进一步下降。变迁 T_{d0}～T_{d2} 表示系统从对应的工作状态进入失效状态的变迁，它们对应的变迁率分别为图 2 的 l_0，l_1，l_2，其中 $l_0 \le l_1 \le l_2$。变迁 T_{up} 表示系统经过修复，从失效状态进入 P_{up0} 状态。

2 不同策略下的软件系统模型

2.1 采用 rejuvenation 的系统

软件老化引起的系统失效是一种突发性的事件，它造成的损失往往是巨大的。为了避免这种情况，在软件运行过程中定时地执行 rejuvenation，主动地停止软件运行，并清理运行环境，然后重启应用软件，使软件运行在良好的环境中，避免意外宕机。对图 3 描述的系统采用 rejuvenation 后的 petri 网模型如图 4 所示。

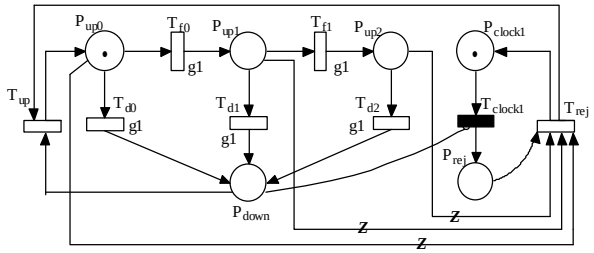


图 4　采用 rejuvenation 的软件系统模型

当定时时间到，变迁 T_{clock1} 触发，位置 P_{clock1} 中的标记消失，P_{rej} 中产生一个标记，变迁 T_{rej} 触发。实施 rejuvenation 时，系统可能处于不同的工作状态，从位置 P_{up0}，P_{up1}，P_{up2} 到变迁 T_{rej} 的相关弧则代表这种情况。Rejuvenation 执行完毕，系统回到初始状态 P_{up0}。从位置 P_{down} 到变迁 T_{clock1} 的禁止弧表示系统失效时禁止定时变迁 T_{clock1} 触发。模型中的 guard 函数（即变迁实施条件函数）g1 表示 rejuvenation 时禁止对应的变迁触发，对应的函数表达式如式（1）。

$$(\#P_{rej} == 1) \ ? \ 0 : 1 \tag{1}$$

2.2 结合 rejuvenation 和检查点的系统模型

虽然执行 rejuvenation 能够阻止软件老化造成的系统性能下降和突发的系统失效，但它是一种重量级的软件避错技术，其成本也是可观的。为了进一步降低成本，我们将检查点和 rejuvenation 结合起来，如图 5 所示：固定 rejuvenation 时间间隔，在 rejuvenation 周期内插入 n 个检查点，每个检查点的时间相同。同时假设程序运行与检查点操作完全串行执行，即检查点操作会使系统处于不可用的状态。

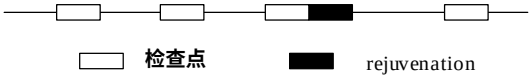


图 5 结合检查点和 rejuvenation 的示意图

由于相对于 rejuvenation 来说，检查点是一种轻量级的软件容错技术，它的成本小于执行 rejuvenation 的成本，所以选取适当的 n 值可以有效的降低系统宕机成本。对图 3 描述的系统采用 rejuvenation 和检查点后的 petri 网模型如图 6 所示。

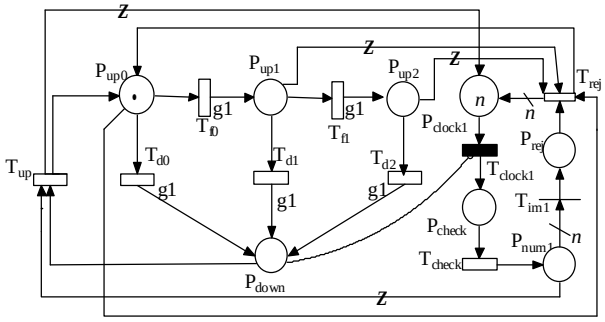


图 6 采用检查点和 rejuvenation 的软件系统模型

和图 4 不同，图 6 的位置 P_{clock1} 中有 n 个标记，当定时变迁 T_{clock1} 触发，将 P_{clock1} 中的一个标记转移到 P_{check} 中，接着变迁 T_{check} 触发，执行检查点，在位置 P_{num1} 中产生一个标记。P_{num1} 相当于一个计数器，记录检查点执行的次数。当检查点执行了 n 次时，P_{clock1} 中的 n 个标记消失，而 P_{num1} 中产生 n 个标记。这时瞬时变迁 T_{im1} 触发，P_{rej} 中产生一个标记，激活变迁 T_{rej}，执行 rejuvenation。从位置 P_{num1} 到变迁 T_{up} 的相关弧表示系统修复时对计数器清零，从变迁 T_{up} 到位置 P_{clock1} 的相关弧表示系统修复时 P_{clock1} 中的标记数恢复为 n。图 6 中其余的位置、变迁、弧和变迁实施条件函数 g1 等表示的含义和图 4 相同。

3 仿真结果及分析

在上一部分，我们给出了在不同策略下系统的 petri 网模型，在此，我们对图 3、图 4 和图 6 进行仿真分析。

假设系统处于失效状态、rejuvenation 状态和检查点状态的概率分别为 P_{down}、P_{rej} 和 P_{check}，对应的每小时系统宕机成本分别为 C_{down}、C_{rej} 和 C_{check}，观察时间段为 T。在不同策略下，系统总的宕机成本表示如式（2）：

$$\begin{cases} cost=P_{down}*C_{down}*T \\ cost_r=P_{down}*C_{down}*T+P_{rej}*C_{rej}*T \\ cost_{r+c}=P_{down}*C_{down}*T+P_{rej}*C_{rej}*T+P_{check}*C_{check}*T \end{cases} \quad (2)$$

其中，$cost$ 代表不采用 rejuvenation 和检查点时的系统总宕机成本，只包括软件失效引起的损失；$cost_r$ 代表采用 rejuvenation 时的系统总的宕机成本，由系统失效引起的损失和执行软件 rejuvenation 引起的损失两部分组成；$cost_{r+c}$ 表示结合 rejuvenation 和检查点时系统总的宕机成本，包括三部分：系统失效引起的损失，执行软件 rejuvenation 引起的损失以及检查点引起的损失。$cost$、$cost_r$、$cost_{r+c}$ 分别对应于图 3、图 4 和图 6。

假设系统的失效修复时间为 30 分钟，成本 C_{down} 为每小时 5000 美元；执行 rejuvenation 操作的平均时间是 10 分钟；检查点操作的平均时间是 2 分钟；观察时间段 T 为 5000 小时。

图 3～6 中对应的变迁率参数设置如表 1 所示（单位：小时$^{-1}$）。

表 1 变迁率参数选择

变迁	变迁率	数值
T_{d0}	l_{d0}	0.004
T_{d1}	l_{d1}	0.00769
T_{d2}	l_{d2}	0.1
T_{f0}	l_{f0}	0.009
T_{f1}	l_{f1}	0.009
T_{up}	l_{up}	2
T_{rej}	l_{rej}	6
T_{check}	l_{check}	30

对图 3 进行仿真，结果表明不采用 rejuvenation 和检查点时的系统总成本 cost 为 106,435.13 美元。图 7 给出了仅采用 rejuvenation 时，当比值 C_{down}/C_{rej} 不同时，rejuvenation 时间间隔变化，即变迁 T_{clock1} 的变迁率变化，对系统总的宕机成本 $cost_r$ 的影响。用 a 来表示 rejuvenation 时间间隔，可知 a 等于图 4 中的 $1/l_{clock1}$。

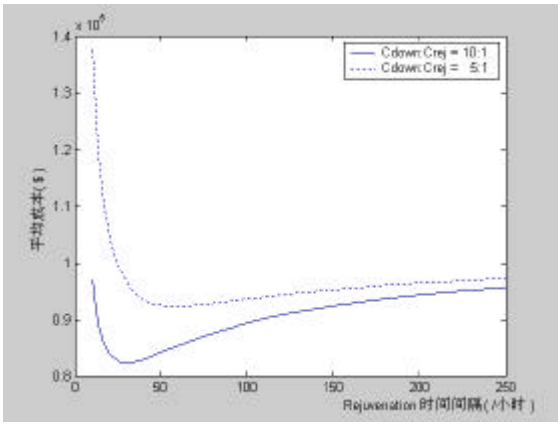
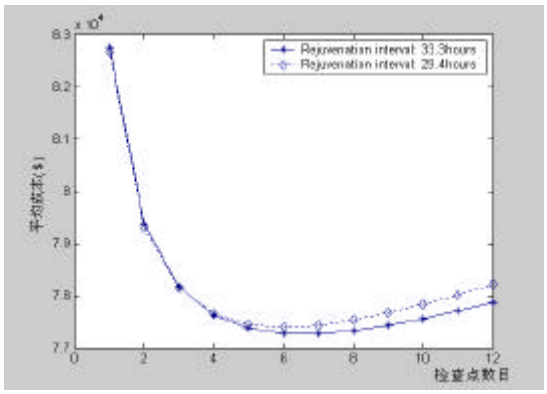


图 7 系统总的宕机成本和 rejuvenation 时间间隔的关系

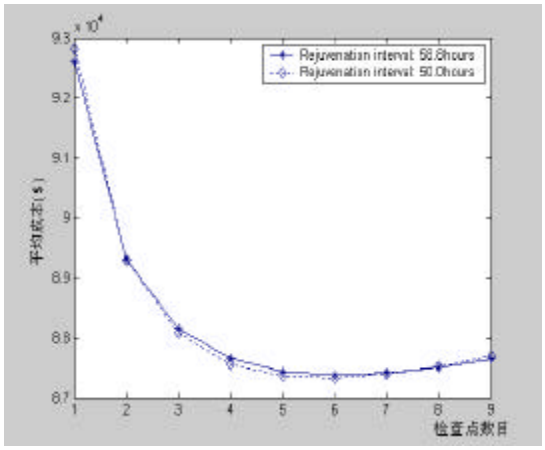
从图 7 可以看出，当 a 较小，即频繁的执行 rejuvenation，系统总是处于不可用的状态，总的宕机成本 $cost_r$ 是可观的；随着 a 逐渐增加，$cost_r$ 会达到一个最小值，此时的 a 就是最优 rejuvenation 时间间隔；当 a 继续增

加，即 rejuvenation 频率下降，系统失效对宕机成本的影响上升，也会使 $cost_r$ 增加。当 C_{down}/C_{rej} 为 10:1 时，最优的 rejuvenation 时间间隔为 29.4 小时，$cost_r$ 为 82,388.15 美元；而当 C_{down}/C_{rej} 为 5:1 时，最优的时间间隔为 58.9 小时，$cost_r$ 为 92,333.27 美元。这是因为 rejuvenation 引起的宕机成本越小，较频繁的执行 rejuvenation 会带来满意的结果。

图 8(a)给出了结合 rejuvenation 和检查点策略时，当比值 C_{down}:C_{rej}:C_{check} 为 100:10:1，rejuvenation 时间间隔固定，插入不同数目的检查点对系统总的宕机成本的影响。用 a 来表示 rejuvenation 时间间隔，用 b 来表示检查点时间间隔，而这时，对应于图 6，b 为 $1/l_{clock}1$，于是 a 等于图 6 中的 $n/l_{clock}1$。从图中可以看出，a 固定为 33.3 小时，最优的检查点数目为 7，总的宕机成本 $cost_{r+c}$ 仅为 77,273.27 美元；a 固定为 29.4 小时，最优的检查点数目为 6，$cost_{r+c}$ 仅为 77,398.07 美元。两者均小于仅采用 rejuvenation 时的总宕机成本 82,388.15 美元。



(a)



(b)

图 8 固定 rejuvenation 时间间隔，
检查点数目对系统总的宕机成本的影响

图 8(b)的含义和图 8(a)类似，所不同的是比值 C_{down}:C_{rej}:C_{check} 为 50:10:1。从图中可以看出，a 固定为 58.8 小时，最优的检查点数目为 6，总的宕机成本 $cost_{r+c}$ 仅为 87,378.89 美元；a 固定为 50.0 小时，最优的检查点数目为 6，$cost_{r+c}$ 仅为 87,324.46 美元。同样，两者均小于仅采用 rejuvenation 时的总宕机成本 92,333.27 美元。

4 结论

软件 rejuvenation 是解决软件老化的一种有效的预防性策略，是软件可靠性研究中的一个新课题。检查点技术是一种广泛使用的软件容错技术。在本文中，我们给出了系统在三种情况下的 petri 网模型：1）既没有采用 rejuvenation 策略也没有采用检查点技术；2）仅采用 rejuvenation 策略；3）结合 rejuvenation 和检查点技术，在固定的 rejuvenation 周期间插入适当数目的检查点，并进行了仿真。

通过仿真实验，我们得到结论：选择合理的 rejuvenation 周期，可以有效的降低存在老化的软件系统的宕机成本；固定 rejuvenation 时间间隔，在其中插入适当数目的检查点，可以进一步降低系统的宕机成本，提升系统性能。

参考文献：

[1] Huang Y, Kintala C, Kolettis N, Funton N D. Software Rejuvenation: Analysis, Module and Applications[C]. Proc. 25th IEEE Int'l Symp. On Fault Tolerant Computing, IEEE Computer Society Pree, Los Alamitos, CA, 1995, 381-390.

[2] Wang Y-M, Huang Y, Vo P, Chung P-Y , C. Kintala. Checkpointing and its applications[C]. In Proc. of Symposium on Fault Tolerant Computer Systems, Pasadena, California, 1995.

[3] Garg S, Moorsel A Van, Vaidyanathan K, Trivedi K S. A Methodology for Detection and Estimation of Software Aging[C]. Proc. 9th Int'l Symp. On Software Reliability Eng, IEEE Computer Society Press, Los Alamitos, CA, 1998, 282-292.

[4] Vaidyanathan K, Trivedi K S. A Measurement-Based Model for Estimation of Resource Exhaustion in Operational Software Systems[C]. In Proc. of the Tenth IEEE Intl. Symposium on Software Reliability Engineering, 84-93, Boca Raton, Florida, November 1999.

[5] Dohi T. Goseva-Popstojanova K, Trivedi K S. Analysis of Software Cost Models with Rejuvenation[C]. Prpc. 5th IEEE int'l symp. High Assurance Systems Engineering, IEEE Computer Society Press, Los Alamitos, CA, 2000, 25-34.

[6] Nicola Victor F , Van Spanje Johannes M. Comparative Analysis of Different Models of Checkpointing and Recovery[J]. IEEE Transactions on Software Engineering, 1990, 16(8).