软件恢复技术研究*)

李 正 万群丽 许满武
(南京大学计算机软件新技术国家重点实验室 南京大学计算机系 南京210093)

Research on Software Rejuvenation

LI Zheng WAN Qun-Li XU Man-Wu
(State Key Laboratory for Novel Software Technology, Department of Computer, Nanjing University, Nanjing 210093)

Abstract Recently, the phenomenon of "software aging", one in which the state of a software system gradually degrades with time and eventually leads to performance degradation or crash/hang failure, has been reported. The aging phenomenon is ubiquitous in all kinds of software systems and hard to eliminate thoroughly. A proactive technique called "software rejuvenation" has been proposed[13] to counter the aging problem and increase the software availability. In this paper, we systematically investigate the basic motivation, concept and development of software rejuvenation, and then, detailedly analyze the main research work on it. At last, a prospect about the future research directions is outlined.

Keywords Software rejuvenation, Software aging, Software fault tolerant, Software availability, Proactive, Preventive maintenance

1. 问题的提出——软件衰老

1.1 软件衰老概念

在最近的很多研究中讨论了"软件衰老"现象。软件衰老是指伴随着软件的运行,系统资源逐渐耗尽或运行错误逐渐积累所导致的系统性能持续下降乃至挂起停机的现象。软件衰老现象是软件可靠性的大敌,有着非常严重的危害作用,在普通的计算中,它可能影响到结果的获取和效率,在战争和国防中,其危害就难以估量了。在文[16]中记述了在海湾战争时,美军正是因为"爱国者"导弹拦截控制系统中的一个软件衰老问题,对伊拉克"飞毛腿"导弹的拦截判断出错,致死美军士兵24人。

软件衰老也是一个很普遍的现象,很多软件系统在持续平稳运行一段时间以后,就往往会出现系统资源大量消耗,本身服务速度、质量下降和挂起停机的现象。常用的服务器,尤其是大访问量、大数据量的服务器,往往每过一段时间就会出现这种现象。由软件衰老所引起的低效、停机和重启,已经见怪不怪了。

1.2 软件衰老的产生原因

软件衰老的主要原因是软件系统中两种 Bug 的广泛存在。

(1) Heisenbug[12] 是与 Bohrbug 相对的一个概念。Bohrbug 是指通过理想情况下的 Debuging 可以排除的 Bug。这类 Bug 是可以简单重现的一当程序运行到该 Bug 所在的位置时就会发生,因此也比较容易在开发和测试直至用户反馈阶段被发现和排除。而 Heisenbug 则不同,Heisenbug 是那种仅当多个特定事件均得到满足的情况下才出现故障的 Bug。比如,一串特定的操作以后的下一个特定操作如果发生时才会出现的 Bug;再比如,多个进程在某些特定情况下才出现的同步性错误。在试图重复执行时,这类 Bug 却不一定会再次发生,因为此时,很多隐含未知的环境条件都已不同了,所以这也被称为"临时性故障"。Heisenbug 具有相当的隐蔽性,几乎不可能被排除。①它们可以很容易通过软件测试阶段一由 Heisenbug 引起的故障,发生条件复杂,而测试阶段不可能模拟一切可能出现的情况;②发生了错误也很难排查原因一根据概率,可能持续运行很长时间才会产生一次故障,而且出现 HeisenBug 的条件是多方面的综合,很难一一确定;③Debug 一般无效一在对程序进行 Debug 的过程中,Debug 操作本身对程序的所附加的改变使得它无法保证这种调试环境下的模拟会与实际的运行环境中的表现绝对一致,这也正是其被称为 Heisenbug 的原因,类似于量子物理学中的海森堡测不准原理。

(2)*致使资源耗竭的 Bug* 编程中的一些缺陷可能会导致资源耗竭的故障,常见的有硬盘交换分区被填满、内存耗尽等等。这些缺陷往往是由内存泄漏或是对用过的资源释放清理不完全而造成的。这是一般编码中常见的问题,但在一般的开发和测试过程中,往往因其不造成程序执行中断等表象特征而很难被完全发现,广泛存在于各种操作系统、中间件和应用程序中。在某些大数据量大负载的实际应用环境下,很容易成为软件衰老的重要原因。

1.3 软件衰老现象难以正向消除

由以上软件衰老现象的成因就可以看出,软件衰老现象是很难消除的。但事情并不如此简单,还有更多的原因,使得消除衰老现象在目前的软件开发方法中几乎是不可能的。比如,有相当部分软件衰老现象的产生原因并不存在于所开发的应用层软件中,而是存在于所购买的成熟商用操作系统和

*)本文受国家自然科学基金(批准号为69973020)和国家863计划课题(课题编号为2001AA111281-2)资助。李 正 硕士生,主要从事软件方法学和程序设计理论方面的研究。万群丽 硕士生,主要研究方向为软件方法学。许满武 教授,博士生导师,主要研究领域为软件方法学,新型程序设计。

中间件等运行支持环境中，而要发现并在应用层软件开发调试中采取某些措施去解决这些问题的难度是非常大的；再比如，在目前流行的基于组件技术的开发中，开发者一般都没有所购买的商品组件的源代码，而这些代码中也很可能存在有可能引起软件衰老的因素。这些都说明了，仅通过控制开发过程、调试过程和测试过程来消除软件衰老现象，是不实际的。

2. 软件恢复技术的产生和概念

2.1 常见的"反应式"软件抗衰机制及其问题

针对软件衰老现象，很多研究者提出了一些技术来尝试解决和减轻这一现象的危害。这些技术基本上均是基于以下想法的某些机制—当监测到由软件衰老引起的软件故障（比如停机现象）发生时，自动化地对软件进程或运行软件的服务器采取重启、重新载入和恢复进度（与检查点技术结合）等手段，从而达到"抗衰"的目的。从本质而言，这是一种"反应式"（reactive）的容错机制。早期，这些方法是在某些特定软件中通过手工编程实现的，不具通用性，后来出现了一些独立的模块，比如 watchd，libft，nDFS/REPL 等[14]，可以方便通用地在软件开发中使用。

基于这种反应式的容错机制，在某些情况下能够提高那些长时间运行系统的系统可用性和数据完整性，它不用人工干预，发现和处理情况比人工更加及时和准确，结合检查点技术可以恢复运行进度。但从另一个方面而言，这种容错机制所可能取得的效果比较有限，有些情况甚至还会引起更多的问题。在从开始衰老直至停机等故障现象发生的整个过程有可能持续很长时间，而其间系统性能、系统可用性、数据完整性都有很大的问题，在这一点上，反应式的容错机制并没有提供任何有效的改善而只是等待；而这个衰老过程中造成的诸多问题很可能遗留下来，在重启以后重新载入的过程中导致各种各样的问题，在有些情况下，可以致使整个自动过程无法完成，而这样可能导致的损失有时比手工干预的过程更大。

2.2 软件恢复技术的基本概念

在这样的背景下，基于另一思路的"软件恢复"技术在1995年被 AT&T 贝尔实验室的 Y. Huang 等人在文[13]中正式提出。

软件恢复是一种"预反应式"（proactive）的容错技术，它主要通过周期性地暂停软件的运行，清除持续运行系统的内部状态、重新启动并恢复为干净的初始或中间状态，抢先防止将来可能发生的更严重的故障。常见的内部状态清理手段有清除缓冲序列，内存垃圾收集，重新初始化内核表、清理文件系统等等，但最常见也是最简单的清理和恢复手段就是重启机器。

因此，软件恢复技术也可以说是一种"抢占式"的预防维护技术，它力求在软件衰老这一渐进的过程中，在软件衰老所引起的故障发生以前的某个时间，抢先清除内部状态，消除可能引起故障的衰老因素，恢复到干净的状态。

软件恢复技术基于一个最基本的设想和前提一按计划地停止并重启应用程序所造成的损失要远小于运行中出现的由软件衰老所引起的故障的损失，而这一点往往是成立的。从应用程序服务停止的时间长短来考虑，有计划地停止和重启，所花费的时间并不多，而因故障在随机情况下的停机，则很可能要过一段相当长的时间才可能被发现（除非有管理者一直不停地监控，而这是很少见的），因此停止服务的时间往往会很长；从停机所造成的代价来考虑，有计划地停止和重启，一般可以控制和安排在系统访问量比较小的时候，这样单位时间造成的损失代价比较小，而随机情况下的停机，则很难确定在何种情况下发生，且往往是在访问量大系统负载重的时候发生，因此单位时间造成的损失代价比较大；从系统服务性能来考虑，软件衰老过程中，系统服务性能往往会比较低，尤其是接近故障发生的时间段，这同样是一种故障损失，而抢先主动重启恢复干净状态，显然在同样时间段里会有更高的服务性能。相较之下，我们可以明显看出基于以上想法的软件恢复技术可以大幅度地减小由软件衰老所引起的故障时间和故障损失。

2.3 软件恢复技术的应用场景

软件恢复技术的使用适用于很多场景，但并不是全部。比如，在个人电脑中也存在着软件衰老的现象，有时候运行软件会不固定地导致死机，有时候出现内存占用导致的资源耗竭，这都是很典型的症状。但软件恢复技术显然不适用于个人电脑，首先，个人电脑使用时一般都由用户在操作，发生情况用户即可及时处理，采用自动恢复反而会造成不便和低效；其次，个人电脑不是长时间地一直运行，而其软件的运行时间更没有规律，不像服务器一般持续运行几种固定服务，运行状态有相当的规律性，因此很难适应这样的技术。

软件恢复适用的场景一般都是要求高可用性或是高数据完整性的服务器软件，比如网络服务器、路由器、大型的 CAD 计算软件、通讯软件系统等等。如果细分，又可以归结为以下两大类：

（1）*长时间运行的计算程序*　在科学计算领域有相当多的软件需要长时间的运行，有的是几天，有的甚至是几个星期乃至一个月以上才能得出最后结果。因此，各种各样的软件衰老因素就成为获取计算结果的极大阻碍，引起运行速度低下、资源耗竭乃至停机不仅降低软件工作效率、延长软件运行时间甚至是失去运行结果不得不反复重来的严重后果。使用软件恢复技术则显然可以解决这个问题。但这里需要强调的一点是，对于自身没有自动保存中间结果和载入进度功能的软件，单纯使用软件恢复技术显然会导致中间结果的丢失而必须重新计算，此时，应该将软件恢复技术和检查点技术结合，利用检查点技术在合适的时候保存干净的中间状态，然后在重启恢复的时候载入中间状态的数据继续运行。

（2）*长时间运行的服务器软件*　目前普遍存在的客户端-服务器结构的软件模式中，服务器端在理想状况下是期望可以持续不断地永久运行的。但由于软件衰老及其他方面的问题，在实际的运行中，这确实只是个理想。由前所述，使用软件恢复技术可以减小服务器停机的时间和代价。与前一类型不同，一般的服务器软件不是长时间运行以后才得出计算结果，而是持续地长时间处理同样类型的操作，即事务处理，比如 Web 服务器就是很典型的例子，因此对中间结果的保存要求不高，强调得更多的是降低低效/停机时间在全部运行时间中的比例。因此，仅使用软件恢复技术即可满足要求。

2.4 软件恢复技术的研究

软件恢复技术所涉及的具体实现并不复杂，根据软件恢复技术的思路和定义，我们可以看到，需要使用的具体技术为"软件中止"，"软件重启"和"状态载入"。这在技术上显然并不困难，而且已经得到了很好的实现，一些相关的可重用的模块和库都已经出现在 UNIX 平台上，可以方便地在具体开发中使用或者是改造已有的软件系统。

所以软件恢复技术真正的研究重点并不在于这些，而是

在于进行软件恢复的时间选择。从软件恢复这种技术被提出至今，几乎所有的研究所关注的问题都集中在软件恢复的时间上。过高频率的软件恢复会增大停机的时间和损失，过低频率的软件恢复则不能保证起到应有的效用，同样的频率和时间间隔下，又应该选择在哪个时间点进行恢复……对于每个不同的系统，我们应该采用什么通用的方法可以得出这些答案呢？

相关的研究者提出很多模型来解决这个问题，并被逐步引入到实际的系统应用中。限于篇幅的关系，我们无法在文中介绍每一种具体的模型的细节和推导，但有关软件恢复技术研究总体的脉络、进展以及各种模型的思路、特点，我们将在下面系统地进行讨论。

3. 软件恢复技术研究的发展

3.1 软件恢复技术研究的重点、标准和思路

软件恢复技术研究的重点基本上完全集中于对软件恢复时间的选择的研究。软件恢复时间的选择方法的优劣，有其唯一的标准，就是在该方法下，软件总停机状态时间（包括软件恢复时的停机、重启时间和仍可能存在的软件衰老所导致的低效、停机、重启状态的时间）在全部运行时间中所占的比例。可以用公式简单表示不使用软件恢复技术和使用软件恢复技术分别如下：

$$Rate_{without\_sf}=\frac{1}{Time_{total}} * Time_{down\_without\_sf}$$

$$Rate_{with\_sf}=\frac{1}{Time_{total}} * (Time_{sf}+Time_{down\_with\_sf})$$

显然，使用软件恢复技术的优势在于后者的分子部分要远小于前者，而且越小越好。

如果再考虑各不同时间段停机代价的不同的话，则应在在此基础上乘以一个基于时间段的代价函数进行估值。而且，在同一时间，没有准备的软件衰老所导致的停机代价与有准备的软件恢复中的停机代价是不同的，前者一般远大于后者，因此它们的代价函数是不同的。可以用公式简单表示不使用软件恢复技术和使用软件恢复技术分别如下：

$$CosRate_{without\_sf}=\frac{1}{Time_{total}} * Time_{down\_without\_sf} * Cost_{aging}(t)$$

$$CostRate_{with\_sf}=\frac{1}{Time_{total}} * (Time_{sf} * Cost_{sf}(t)+Time_{down\_with\_sf} * Cost_{aging}(t))$$

（上式 L * C 中，C 往往是时间的函数，因此 L * C 实际应该是积分，上述表达式仅为示意表达式。）

由此可见，软件恢复技术研究主要是关注于寻找最优的软件恢复时间和频度的计算模型和方法。有相当多的研究者已经开始这方面的尝试，但总的来说，大都可归为两条不同的思路—基于时间的研究思路和基于测量的研究思路。

3.2 基于时间的研究思路

基于时间的研究思路出现得比较早，在软件恢复概念刚被提出的论文[13]中，所采用思路基本可算是基于时间的。基于时间思路的研究其根本在于，对于软件系统的状态关系建立模型，用数学方法求解得出软件恢复时间的最优值与各参量之间的函数关系，而参量的值可以通过实际系统一段时间的试验或实际运行而得到，这样软件恢复时间和频度等就可以得解。基于时间的方法的重点往往在于利用随机 Petri 网等工具对软件系统的状态和转换进行刻画建模，结合随机过程等数学方法求解，最后得出在该模型下的最佳恢复时间的公式或关系，并结合具体实例进行验证。

Y. Huang 等在初次提出软件恢复技术概念的文[13]中，采用了一个基于连续时间马尔可夫链的简单模型。他将软件的正常执行过程分为三态（如图1）：启动后的一段高度健壮态 S_0，发生衰老和停机的概率基本为0；经过一段时间的运行，进入衰老可能态 S_P，在该状态下随时可能发生衰老和停机现象；当在衰老可能态停留一段时间以后，就会进入故障停机态 S_F；在故障停机态等待重新启动又需要相当长的时间，然后恢复到高度健壮态 S_0。各态之间的平均转换速度分别为 r_1、r_2和 λ。然而当引入软件恢复技术，则需要在原来的基础上修改状态转换图（如图2），在进入衰老可能态 S_P 以后，出现两条路经，一条是通向故障停机态 S_F，一条是通向软件恢复态 S_R，软件恢复态与故障停机态一样，最后都是因重启而重新进入高度健壮态 S_0，各态之间的平均转换速度分别为 r_1、r_2、r_3、r_4和 λ。根据以上的模型，可以通过列出状态间的平均转换速度和各状态的概率分布关系的等式，最后分别得出使用和不使用软件恢复技术情况下的停机时间和代价与各状态间的转换速度之间的函数关系。因为其他各状态间转换的速度均为通过测量得到的常量，仅有从衰老可能态 S_P 到软件恢复态 S_R 的速度是可控的变量，所以可以简单得出是否使用软件恢复技术的临界条件，以及使时间和代价函数取得最小值的条件。

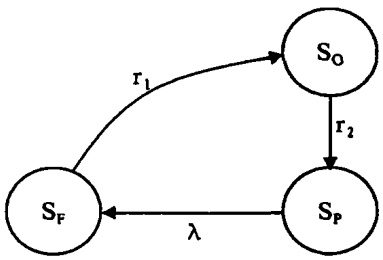

图1 概率状态转换模型（未使用软件恢复技术）

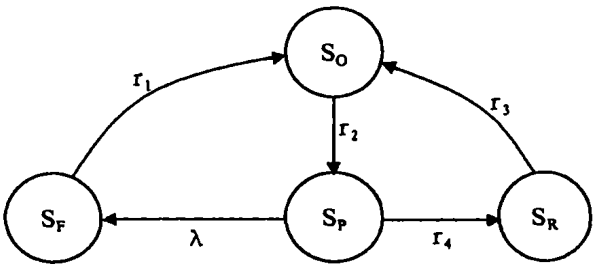

图2 概率状态转换模型（使用软件恢复技术）

这是一个非常理想化的模型，实际的运行状态不可能完全是这样简单。在某个计算出的应该执行软件恢复的时刻，很可能软件已经停机了，或者尚未进入衰老可能态，那么这个状态图就完全不同。原因很简单，我们这里所取的状态转换速度都是平均值，而每次具体执行的时间却肯定是不同的。但也正因为所取的是平均值，从统计的观点而言，这个粗糙的模型仍然可以取得不错的效果，Y. Huang 在文[13]中提供的一些实际系统测试的例子说明了这一点。

S. Garg 等在 Y. Huang 的定义和模型基础上开始研究更细致的模型[8]，他们采用马尔可夫再生随机 Petri 网模型来描述和进行定量研究。在这个更细致的模型中，我们可以看到，与文[13]中的简单模型不同，时间与状态转换的所有可能关系均得到了很好的刻画，我们前面所说的问题得到了解决。

这篇文章同时确立了计算连续两次软件恢复操作之间所应取的时间间隔——"软件恢复间隔"的思路。后来在这个方向的研究莫不与此类似——提出概念，为时间和状态转换建模，利用数学工具计算时间的临界解和最优解，通过计算和实例来说明这些模型和概念的实际效果。

S. Garg 在完成文[8]的研究后不久，与 Y. Huang 合作，在原有的模型基础上进行了进一步的改进，发表了文[6]。在文[8]的模型中并没有详细考虑不同时间点的系统负载情况与软件恢复时间选择之间的关系，而文[6]中的改进恰是针对于此。负载量在大部分情况下是正比于该时刻的系统的事务处理量或访问量的，在实际的系统中，在不同负载情况下的停机，其停机代价往往是不同的，负载大的情况下，一般所接受和处理的业务量大，停机的损失会更大。因此，要考虑不同时间停机代价的不同，系统负载随时间的分布关系正是首要的考虑。S. Garg 等在文[6]中提出了采用随机回报网来刻画系统。随机回报网与标准 Petri 网和大多数的随机 Petri 网不同，具有弧权变量、变迁实施函数和变迁实施优先级等特性，因此非常适合用来刻画考虑系统负载的条件下的状态转换。同时，在这个模型下，判断最优解的标准可以是根据时间的，也可以是根据负载的，当然最全面的选择是同时基于这两者的标准。在文章的最后，作者提供了一些实验结果作为说明。

在接下来的研究中，研究者开始将已发展的理论进一步具体化、方法化。S. Garg 与 Y. Huang 在前述理论发展的基础上，将软件恢复技术和检查点技术结合[7]，具体探讨了如何使用这两种技术最小化一个应用程序的完成时间的问题，这正是我们在第三部分中提到的第一种类型场景的应用；在文[17]中，A. Pfening 提出了一个基于马尔可夫决策过程的框架和方法，基于不同的策略决定软件恢复的时间最优值，与文[7]不同，这篇文章所研究的对象正是在第三部分中所提到的第二种类型场景的应用——事务处理型应用软件系统，其实际的应用背景是 AT&T 的通讯系统。

在1997年发表的文[9]中，S. Garg 等开创性地提出，系统处理低效状态与挂起停机状态同样是软件衰老的重要体现，尤其在面向事务处理型的应用中，刻画和计算系统低效状态并以此为判断根据应该要比偶然发生的停机状态更加精确和合理。这一点在前面的各种模型中都是没有的，衰老过程往往简单刻画为从衰老可能态到故障停机态的单一状态转换描述。他采用了一个非马尔可夫模型对基于事务处理型应用软件系统进行研究并以数据说明。

表1 模型比较

	衰老刻画		故障分布的一般性?	刻画了衰老与负载的关系?	判断策略根据		
	故障停机	系统低效			可用性（时间）	业务损失率(代价)	响应时间
[13]	✓				✓		
[8]	✓				✓		
[6]	✓				✓	✓	
[7]	✓		✓				
[17]		✓	✓			✓	
[10]	✓	✓	✓	✓	✓	✓	✓

在1998年的 IEEE Transactions on Computers 上，S. Garg 等[10]综合前述理论成果，提出了一个最为完善的综合模型，涵盖了前述理论发展中所有的因素(参见表1)。

这是一个综合的最具通用性的模型，几乎所有前述模型均可视为这个模型的特例。除此之外，这个模型还考虑和刻画了更多的内容。比如，对衰老现象发生与负载的内部关系的刻画。有相当多的研究表明，临时性故障(软件衰老的主要原因之一)的发生部分地是由过载情况所引起的。这在之前的模型中没有对应的函数刻画，而在这个模型中，允许故障和低效状态是时间、即时负载和累计平均负载的函数。再比如，在计算和判定最优化软件恢复时间时，这个模型也新增了基于响应时间的判断策略，这在某些有此类特定要求的系统中非常适用。

文[10]是基于时间思路的软件恢复技术研究迄今为止最重要的阶段性的成果。

3.3 基于测量的研究思路

在基于时间的方法发展到一定阶段以后，关于软件恢复技术的研究出现了另一分支——基于测量的思路。其主体在于，在软件运行的实际过程中，监视和测量系统的运行状态的各项性能参数，再根据软件本身的一些参数，动态地进行软件衰老的速度、时间、可能性及软件恢复代价等的计算评估，作出是否与何时采取软件恢复措施的决策。基于测量的思路较之基于时间的思路而言，实践性要更强一些，在其最开始的时候就是以具体的实用方法的形式出现。

S. Garg 等在文[11]中第一次提出了基于测量思路估测软件衰老情况的方法。主要内容是，通过对某些容易产生衰老现象而导致低效或停机现象的系统资源进行监视，用统计学模型估算和验证软件衰老的情况，根据不同的策略决定是否以及何时采取软件恢复措施。在具体的系统设计中，使用基于 SNMP 的分布式监测工具从系统中获取关键参数，常见的比如内存使用情况、交换区使用情况、CPU 占用率、空余通讯通道数等等，这些都可能是软件衰老的重要因素。为了定量比较系统中各种不同资源的消耗，提出了统一使用"预期耗竭时间"并作为公共尺度，并采用统计学中斜率估值技术来对不同的系统参数进行计算。论文最后在5台运行不同服务 UNIX 工作站上的实验结果进行了详细的分析和讨论。

在 S. Garg 等[11]对于资源消耗速率的估测是仅与时间相关的，而 K. Vaidyanathan 等则在此基础上更进了一步[21]，将系统的工作负载状态同时考虑在内一资源消耗的速率既是时间的函数又是系统负载的函数。这是非常合理的优化，很多研究已经表明了软件衰老与系统负载之间的紧密联系，这样的改进使得对于系统可以更精确地刻画和判断。在文[11]中所使用的数学模型是基于马尔可夫回馈模型和半马尔可夫回馈模型。

A. Bobbio 等的研究提出了一种细粒度模型来平滑软件衰老过程[2]，与前述研究将软件运行状态简单地分为三态不同，他们提出的模型将性能下降的全过程分为很多个阶层，其前提是能够通过对系统参数的测量和计算确定当前系统状态

所处的阶层，并据此，通过计算产生对应于该层的软件恢复决策。文章的具体实验背景是基于事务处理型数据库系统，取得了相当好的效果。

L. Li 等[15]提出了一种基于时序分析的方法，来测量和估算 Web 服务器的软件衰老的情况。这项研究以 Internet 常用的 Linux 系统和 Apache 服务器平台为研究实例。在具体系统设计中，首先通过系统工具监测和收集 Web 服务器的记录资源使用的情况和一些活动参数，然后根据这些数据构造出时序 ARMA 模型来计算软件衰老和资源消耗的情况。实验时通过程序人工模拟对服务器的访问流，其结果表明这套方法有着很好的实际效果，而且计算强度较小，相当高效。

3.4 两种研究思路的比较

基于时间的和基于测量的研究思路各有其特点。总的而言，一者基于宏观，一者基于微观；一者关心外部特征，一者关心内部特征。基于时间的方法是基于宏观的方法，说它宏观，是因为基于时间的思路中并不关心究竟是哪种因素造成软件衰老的发生，究竟是 CPU 的占用还是内存泄漏或是进程表的问题，这都是不管的，所关心的只是外部表现出的规律，即软件衰老现象整体的发生和分布情况。而基于测量的方法是基于微观的，它具体测量和估算可能引起软件衰老的各种因素及其变化趋势，并由此综合决定是否与何时采取软件恢复措施，因此关心的是内部特征。

正因为这两种思路的不同特性，它们必然带有各自固有的优势和劣势。

基于时间的方法比较宏观，考虑整体特性，因此在软件运行状况变化不大的情况下（比如某些访问量和负载变化不大的系统中），通过一次计算的结果确定软件恢复时间和间隔，就可以长期地简单使用，不像基于测量的方法，在运行时必须不断地测算。但在软件的运行和负载经常发生大的变化的场景中，基于时间的方法就不那么适用了。基于测量的方法基于微观，因此往往要比基于时间的方法更精确也更灵活一些，它实时地判断情况，适用于情况变化随机性相对大一些的场景，而基于时间的方法适用于情况变化随机性小一些或者变化比较慢的场景。

偏重微观和内部特征的特点同样给基于测量的方法带来了难以克服的缺点。首先，基于测量的思想是分别对于每一种可能引起软件衰老现象的资源进行监测，而据此对是否和何时采取软件恢复进行决策。但这并不总是某些软件衰老情况的实际反映，比如某些衰老并不是单个资源问题在起作用并产生状态转换，而是多个资源问题在互动地起作用和产生状态的转换，那么基于各个资源单独的测量和趋势分析就不再是合理的，而考察基于所有资源的全部可能关系又显然是不可能的；再比如，很可能存在某种衰老的因素目前不为我们所知，也不在我们所监控的这些资源当中，或者是某些资源即使知道也很难监视，这种情况下，基于测量的方法也显然会失效。而此时，基于时间方法的宏观特性的优势就显现了出来，它不管系统内部致使衰老的因素都有哪些、如何相互作用，它只关注外部特征的规律性。

基于测量的方法还有一个缺点在于，它需要不断监视软件系统运行的状态并进行估算，如果不是在分布式系统中——即监视工作也需在本机完成的话，这会增加系统的负担，降低系统的效率，其实也相当于延长了系统的执行时间。在重点考虑效率的场景中，也相当于相对增加了一些停机时间在全部运行时间中的比例，这是一个不可忽略的因素。

因此，基于这两种思路的方法各有各的特点和用处，很难说哪一个更有效，要根据具体应用的特点而定。目前，这两种思路的研究都在进一步地深入。

3.5 新的研究进展

前面回顾了两种基本研究思路的发展脉络，一些最近的研究往往是在此基础上或是作出了一些创新的改进或是讨论将软件恢复技术应用到实际场景中的一些问题。

A. Bobbio 等[1]首次讨论了将软件恢复，软件载入和检查点三种技术集成到一个应用中的问题，采用了基于时间思路的流体随机 Petri 网模型。K. S. Trivedi 等[18]概述了软件恢复技术理论方面的发展，重点对基于时间思路的研究进行了一定的总结。在这篇文章中，第一次在同一个应用中对基于时间的方法和基于测量的方法进行了实验比较，并得出一些有意义的结论。在文[4,5]中，T. Dohi 等在 Y. Huang[3]的模型的基础上，提出了一种基于 TTT(time on test transform) 的无参量统计算法，使用该算法可以不考虑故障的时间分布，而故障时间分布的获取在一般的基于时间方法的具体应用中是必须且非常麻烦的。因此在某些实用场合中，T. Dohi 提出的算法更具实践性。V. Castelli[3]和 K. Vaidyanathan[19]，则探讨了将软件恢复研究发展的成果运用到集群计算机系统中的方法，其成果是已经集成到的 IBM 的 x 系列服务器集成管理框架中的软件恢复 Agent (SRA)。

K. Vaidyanathan 等[20]提出的观点和方法是近期的一项重要发展，指出了未来的某些可能的发展方向。它以基于时间的思路为主体，引入基于测量的方法中的一些元素，提出了一种基于视察的方法。它的主体思想是，首先使用基于时间思路中的经常使用的建模方法来计算软件运行状态的大致分布，据此进一步计算某些需要观测的关键点的时间和间隔，在系统运行时，在这些关键点进行观测系统状态并根据当前软件衰老状态所处阶段决定采取的不同的维护类型。这样就将基于时间的和基于测量的方法有机地结合起来，它既不像传统的基于时间的方法没有实时的观测而不够精确和缺乏灵活性，又不像基于测量的方法有时不能正确反映软件衰老的原因和趋势，在相当程度上避免了二者的一些缺点。

结束语 软件恢复技术作为一种预反应式的软件容错机制，产生时间并不长，但已日益显现出其重要价值。AT&T 电信系统很早便开始了这方面的实用研究，IBM 也开始在它的商用服务器中预装了包含软件恢复技术的集成工具，美国国家软件委员会主席 Larry Bernstein 说，“我希望所有的软件系统都可以配备这种技术”。我们有理由相信，软件恢复技术将在未来的软件系统容错性的研究和发展中占据越来越重要的地位。

本文系统地论述了软件恢复技术的成因、起源和概念，重点介绍并分析了软件恢复技术研究发展的主要思路、方法和现状，希望对国内在该领域的研究起到一定的推动作用。在最后，我们对软件恢复技术未来的研究方向作出以下一些简单的分析和预测，以供参考。

(1)基于测量思路的研究刚刚起步，并有非常广阔的空间，相信会在未来的研究中占据更大的比重；

(2)将产生更多同时结合基于时间和基于测量两种思路的研究；

(3)结合具体应用的实证性研究将进一步增加；

(4)在软件容错领域，软件恢复中的概念和其他技术中的概念结合，将产生更多的新概念和新思想。

参考文献

1 Bobbio A, Garg S, Gribaudo M, Horvath A, Sereno M, Telek M. Modeling Software Systems with Rejuvenation, Restoration and Checkpointing through Fliud Stochastic Petri Nets. In: Intl. Conf. on Petri Nets and Performance Models, PNPM99, Sep. 1999

2 Bobbio A, Sereno M, Anglano C. Fine Grained Software Degradation Models for Optimal Software Rejuvenation Policies. Performance Evaluation, 2001, 46: 45～62

3 Castelli V, et al. Proactive Management of Software Aging. IBM Journal of Research & Development, 2001, 45(2)

4 Dohi T, Goseva-Popstojanova K, Trivedi K S. Analysis of Software Cost Models with Rejuvenation. In: IEEE Intl. Symposium on High Assurance Systems Engineering, HASE 2000, Nov. 2000

5 Dohi T, Goseva-Popstojanova K, Trivedi K S. Statistical Non-Parametric Algorithms to Estimate the Optimal Software Rejuvenation Schedule. In: Pacific Rim Intl. Symposium on Dependable Computing, PRDC 2000, Dec. 2000

6 Garg S, Huang Y, Kintala C, Trivedi K S. Time and Load Based Software Rejuvenation: Policy, Evaluation and Optimality. In: First Fault Tolerance Symposium, FTS-95, Dec. 1995

7 Garg S, Huang Y, Kintala C, Trivedi K S. Minimizing Completion Time of a Program by Checkpointing and Rejuvenation, ACM SIGMETRICS 1996, May 1996

8 Garg S, Puliafito A, Telek M, Trivedi K S. Analysis of Software Rejuvenation using Markov Regenerative Stochastic Petri Nets. In: Int'l. Symp. on Software Reliability Engineering, ISSRE 1995, Oct. 1995

9 Garg S, Puliafito A, Telek M, Trivedi K S. On the Analysis of Software Rejuvenation Policies. In: Annual Conf. on Computer Assurance (COMPASS), June 1997

10 Garg S, Puliafito A, Telek M, Trivedi K S. Analysis of Preventive Maintenance in Transactions Based Software Systems. IEEE Transactions on Computers, Jan. 1998

11 Garg S, van Moorsel A, Vaidyanathan K, Trivedi K S. A Methodology for Detection and Estimation of Software Aging. In: Int'l. Symp. on Software Reliability Engineering, ISSRE 1998, Nov. 1998

12 Gray J, Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers, San Mateo, CA, 1993

13 Huang Y, Kintala C, Kolettis N, Fulton N. Software Rejuvenation: Analysis, Module and Applications. In: IEEE Intl. Symposium on Fault Tolerant Computing, FTCS 25

14 Huang Y, Kintala C M R. Software Implemented Fault Tolerance: Technologies and Experience. In: Proc. of 23rd Intl. Symposium on Fault-Tolerant Computing, Toulouse, France, June 1993. 2～9

15 Li L, Vaidyanathan K, Trivedi K S. An Approach for Estimation of Software Aging in a Web Server. In: Intl. Symposium on Empirical Software Engineering, ISESE 2002, Nara, Japan, Oct. 2002

16 Marshall E. Fatal Error: How Patriot Overlooked a Scud, Science, Mar. 1992. 1347

17 Pfening A, Garg S, Puliafito A, Telek M, Trivedi K S. Optimal Software Rejuvenation for Tolerating Software Failures. Performance Evaluation, 1996, 27 & 28,

18 Trivedi K S, Vaidyanathan K, Goseva-Popstojanova K. Modeling and Analysis of Software Aging and Rejuvenation. In: IEEE Annual Simulation Symposium, April 2000

19 Vaidyanathan K, Harper R E, Hunter S W, Trivedi K S. Analysis and Implementation of Software Rejuvenation in Cluster Systems. ACM SIGMETRICS 2001/Performance 2001, June 2001

20 Vaidyanathan K, Selvamuthu D, Trivedi K S. Analysis of Inspection-Based Preventive Maintenance in Operational Software Systems. In: Intl. Symposium on Reliable Distributed Systems, SRDS 2002, Osaka, Japan, Oct. 2002

21 Vaidyanathan K, Trivedi K S. A Measurement-Based Model for Estimation of Software Aging in Operational Software Systems. In: Int'l. Symp. on Software Reliability Engineering, ISSRE 1999, Nov. 1999

22 林闯. 随机 Petri 网和系统性能评价. 清华大学出版社, 2000

(上接第49页)

13 Lunt T F, et al. IDES: The enhanced prototype, A real-time intrusion detection system: [Technical Report SRI Project 4185-010, SRI-CSL-88-12]. CSL SRI International, Computer Science Laboratory, SRI Intl. 333 Ravenswood Ave., Menlo Park, CA 94925-3493, USA, Oct. 1988

14 Lunt T, Jaganathan R, Lee R, Whitehurst A, Listgarten S. Knowledge-Based Intrusion Detection. In: Proc. of the 1989 AI Systems in Government Conf. March 1989

15 Miller B P, et al. A Re-examination of the Reliability of UNIX Utilities and Services: [Technical Report]. Department of Computer Science, University of Wisconsin, 1995

16 Porras P A, Neumann P G. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In: 19th National Information System Security Conf. (NISSC), 1997. http://www.csl.sri.com/emerald/emerald-niss97.html

17 Sebring M M, et al. Expert systems in intrusion detection: A case study. In: Proc. of the 11th National Computer Security Conference, Baltimore, Maryland, NIST, 1988

18 Sielken R. Application Intrusion Detection. University of Virginia Computer Science: [Technical Report CS-99-17]. June 1999

19 Sielken R, Jones A. Application Intrusion Detection Systems: The Next Step. ACM Transactions on Information and System Security, Submitted 1999

20 Smaha S E. Haystack: An intrusion detection system. In: Proc. of the IEEE Fourth Aerospace Computer Security Applications Conf. Orlando, FL, USA, December 1988. IEEE, IEEE Computer Society Press, Los Alamitos, CA, USA

21 Snapp S R, Brentano J, Dias G V, et al. DIDS (Distributed Intrusion Detection System)- Motivation, Architecture, and An Early Prototype. In: Proc. of the 14th National Computer Security Conference, Oct. 1991

22 Snapp S R, Smaha S E, Teal D M, Grance T. The DIDS (distributed intrusion detection system) prototype. In: Proc. of the Summer USENIX Conf. San Antonio, Texas, USENIX Association, 1992. 227～233

23 Vigna G, Eckmann S T, Kemmerer R A. The STAT Tool Suite. In: Proc. of the 2000 DARPA Information Survivability Conf. and Exposition, 2000

24 Vigna G, Kemmerer R. NetSTAT: A Network-based Intrusion Detection Approach. In: Proc. of the 14th Annual Computer Security Application Conf. Scottsdale, Arizona, Dec. 1998