学 术 论 文

计算系统多级抗衰技术研究

游　静[1,2]，徐　建[2]，李千目[2]，刘凤玉[2]
（江苏工业学院计算机科学与工程系，江苏常州　213164；南京理工大学计算机系，南京　210094）

［摘要］　为了对抗软件老化，提出一种细粒度的、事前的、主动的多级软件抗衰技术。通过分析系统资源的占用和损耗情况，判定系统性能的衰退规律，并据此制定了基于时间的多级软件抗衰策略和基于检测的多级软件抗衰策略，采用有限状态自动机对两种策略进行形式化描述，最后通过 Web 服务案例说明策略的制定过程，给出仿真结果。实验表明，较之单一系统级软件抗衰，多级抗衰策略可以进一步缩短 MTTR，提供更高的系统可用性，同时将抗衰成本降到更低。

［关键词］　软件抗衰；软件老化；系统可靠性；系统可用性；自动机
［中图分类号］ TP302.7　　［文献标识码］ A　　［文章编号］ 1009－1742（2007）02－0036－08

在过去的 20 年里，软件系统的性能有了上万倍的提高，但随之而来的软件复杂性也使其在推出之前不可能做全面的测试，异常和可能导致误操作的程序容易被忽视，导致程序在连续运行过程中会出现内存的占用和泄露、未释放的文件锁、数据更新不及时、存储空间碎片以及舍入误差的累积等不良情况，软件性能随之衰退，并最终导致软件的失效，这种现象称为软件老化（software aging）[1~3]。

模块化的软件设计方法使失效的系统越来越难以维护和操作，庞大的用户群意味着系统失效会带来更大的损失，因此软件老化对系统的可靠性和可用性构成巨大威胁。针对软件老化现象，特别提出了一种系统资源耗尽前的重启技术——软件抗衰（SR，software rejuvenation）[1,3~5]。SR 是对软件老化现象的积极反应，即当软件性能衰退到一定程度时，终止程序的继续运行，重启系统以清理其内部状态（如进行垃圾收集、刷新操作系统内核表、重新初始化内部数据结构等），从而释放操作系统资源，使软件性能得到恢复[1,3]。这种技术不仅有助于延长 MTTF（mean time to failure），使系统运行时间更长，而且失效前的重启消耗的时间也较少，缩短了 MTTR（mean time to repair），从而大大提高了系统的可靠性和可用性。

为了提高系统的可靠程度，还开展了许多其他缩短 MTTR 的研究。如 UC Berkeley 和 Stanford 大学的合作项目——面向恢复的计算（ROC，recovery oriented computing）[6,7]，George Candea 提出的微重启（Microreboot）[8,9] 等。这些技术都强调许多已知和未知的软件故障可以通过软件重启排除，并通过软件的部分重启来缩短 MTTR。ROC 中的递归重启技术（RR，recursive restartablity）首先分析软件各模块之间的关系，按级别建立重启树，如果软件出现故障，从重启树最底层的模块开始重启，不能恢复则上推一级，执行更大范围的重启；Microreboot 则希望将重启执行到更细的粒度，并强调软件各功能模块间的松耦合，适合于按松耦合原则开发的软件[10]。

多级软件抗衰技术（MSR，multilevel software rejuvenation）是针对难以检测和避免的软件老化错误导致的资源损耗和系统失效，在 SR 的基础上，

［收稿日期］　2005－09－27；修回日期　2005－11－10
［基金项目］　国家自然科学基金资助项目（60273035）；国防科工委基础应用项目（K1704060511）
［作者简介］　游　静（1975－），女，河北石家庄市人，博士，江苏工业学院讲师

借鉴 Microreboot 和 RR 的思想提出一种细粒度的、事前的、主动的恢复技术。这里的资源损耗并不是指资源的占用量，而是指占用后不使用也不释放的资源量。通过分析系统资源（如 CPU，内存，页面空间等）的占用和损耗情况，并预先确立软件各功能模块间的关系，按系统的衰退规律或系统资源的实时检测值确定抗衰粒度和重启对象，制定相应的抗衰策略，采用有限状态自动机（FSA，finite-state automaton）对多级抗衰策略进行形式化描述，最后通过 Web 服务案例分析系统资源的衰退规律，说明策略的制定过程，并给出仿真结果。

1　系统性能衰退规律分析

资源消耗趋势估计是衰退规律分析的前提，但使用软件衰退的监控和采集工具所获得的时间序列数据中不可避免地存在噪声，因此在进行趋势估计之前，有必要对原始数据进行预处理，去掉噪声数据。采用 Haar 小波平滑技术完成数据预处理，通过线性拟合进行趋势估计，通过分析资源的占用率、损耗率等参数确定性能衰退规律。

1.1　基于 Haar 小波变换的数据平滑

假设 $\{X_t, t=1, \cdots, N\}$ 是原始的时间序列，基于小波变换的时间序列分析是在不同的分辨层或不同的时间尺度下，把序列分解成尺度函数 ϕ_{jk} 和小波函数 φ_{jk}，其中 $\phi_{jk}=2^{-j/2}\phi(2^{-j}t-k)$ 是原序列的近似，$\varphi_{jk}=2^{-j/2}\varphi(2^{-j}t-k)$ 展示一些细节信息，则 $X(t)$ 写成

$$X(t)=\sum_{k\in Z}a_{j,k}\phi_{j,k}+\sum_{j\leqslant J}\sum_{k\in Z}d_{j,k}\varphi_{j,k}(t),$$

式中　$a_{j,k}=(a_{j-1,2k}+a_{j-1,2k+1})/2$，

$d_{j,k}=(a_{j-1,2k}-a_{j-1,2k+1})/2$。

为了简化计算通常取 $a_{0,k}=X_k$，$k=1, \cdots, N$；J 是时间序列的最大分辨层数，按 $J\leqslant \ln 2^N$ 计算。

为了去除噪声数据，考虑 2 个时间序列数据 $A=\{X_t, t=1, \cdots, N\}$ 和 $B=\{X_{t+1}, t=1, \cdots, N\}$，确定阈值

$$\tau_j=\sigma_j(2\ln n_j)^{1/2},$$

式中 σ_j 代表标准偏差的估计，n_j 代表在尺度 2^j 上的小波系数 $d_{j,k}$ 的个数。若 $|d_{j,k}|\leqslant\tau_j$，则该点为噪声数据，使用当天均值将其平滑掉。

1.2　衰退规律分析

对任一被损耗的系统资源，用以下 3 个变量来衡量资源的占用情况、损耗程度及实施重启的有效程度：

$U_R=[$(资源占用总量)/(资源总量)$]\times 100$（%）；

$W_R=[$(资源消损耗总量)/(资源占用总量)$]\times 100$（%）；

$B_m=[$(模块 m 导致的资源消损耗量)/(重启模块 m 引发的停机时间)$]\times 100$（%）。

通过软件运行过程的监控或经验数据的分析，可知各种系统资源的损耗对系统性能衰退的影响程度，并确定应该按哪种或哪几种资源的损耗来确定抗衰策略。

约定：当某种系统资源的占用率 U_R 大于预定极限值 U_L，且资源的损耗率 W_R 大于预定极限值 W_L 时，才考虑软件的抗衰；将 B_m 值最大的模块作为抗衰策略的重启对象。其中 $0<U_L$，$W_L<1$，由系统管理者根据经验确定。因为当系统尚有足够的资源可以使用时，即使 W_R 较高，也不会对系统整体性能产生太大影响；而当某种资源的占用率很高，系统已无足够资源可用，但 W_R 却很低时，说明系统的性能瓶颈不是资源损耗，而是计算机的处理速度等其他因素。

1.3　概率分布函数确定

软件老化过程中系统至少具有 2 个状态：健壮状态和失效状态。在制定软件抗衰策略前必须已知健壮状态转移到失效状态的概率分布。通过经验数据的统计分析或多次实验结果可得到离散概率序列 $\{p_k, 0\leqslant k\leqslant N\}$：

$p_k=[$(从健壮状态进入失效状态的观测点数)/(该时刻的观测点总数)$]\times 100$（%）。

对该序列进行曲线拟合可以得到状态转移的近似概率分布函数，据其可进行失效概率的分析和重启时间的预测，制定合适的软件抗衰策略。

2　制定多级软件抗衰策略

与 SR 一样，通常将 MSRP 分为两类：基于时间的多级软件抗衰（TMSRP，time-based multilevel software rejuvenation policy）和基于检测的多级软件抗衰策略（DMSRP，detection-based multilevel software rejuvenation policy）。

2.1　重启群

当系统的一个模块 A 重启时，可能导致其他模块 B 的故障或错误，此时称模块 B 对模块 A 是

重启相关的，即当 A 重启时，B 同时重启。两模块是否重启相关取决于其间连接的紧密程度，即耦合性。为了保证在重启前后系统状态和数据的一致性，做如下定义：

定义 1　若模块间存在内容耦合和公共耦合，则两模块是相互重启相关的，必须同时重启。

定义 2　若模块间存在控制耦合，则被控制模块对控制模块是重启相关的，控制模块重启时，被控制模块必须同时重启，反之不然。

定义 3　若模块间只存在数据耦合或非直接耦合，则两模块是重启无关的，可独立重启。

在实施抗衰策略前，需测知导致系统性能衰退的模块，并求得其所有重启相关模块。

定义 4　所有必须与某一模块同时重启的模块构成该模块的重启群，用 $S[\cdot]$表示。

以对象式程序设计方法开发的某软件的体系结构（图 1）为例求各模块的重启群。

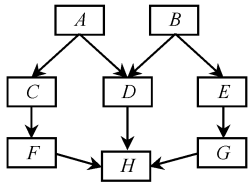

图 **1**　平面体系结构（有向无环）

Fig.1　Plate architecture

该程序设计方法将数据和操作封装在一起成为模块，参数的传送通过模块间直接交换消息实现，因此模块之间最紧密的耦合是控制耦合。考虑最不理想的情况，即各模块间关系均为控制耦合，箭头前端模块总被尾端模块调用，则可推断各模块间的重启相关性，求得任一模块的重启群。如 $S[A]=\{A, C, D, F, H\}$，$S[E]=\{E, G, H\}$，$S[H]=\{H\}$。

2.2　制定 **TMSRP**

TMSRP 表现了多级重启的有序性和嵌套性：前一级重启嵌套在后一级重启中，当前一级重启不能将系统性能恢复到预定值时执行后一级重启，最后执行整个系统的重启。按上述约定，对于具有稳定衰退规律的软件，可获得重启级别和所有重启对象。

对图 1 所示软件体系结构，按 2.1 节方法对其运行时消耗的系统资源进行分析，可得知导致系统性能衰退的主要因素如物理内存的损耗，且主要损耗模块为 G，E，B，分析各模块重启群，可知 $S[G]=\{G, H\}$，$S[E]=\{E, G, H\}$，$S[B]=\{B, D, E, G, H\}$。由此可得各级重启对象：

1）比较各重启群重启时单位时间可释放的物理内存量，将释放量最多的重启群作为第一级重启对象。如本例中 $B_{S[E]} > B_{S[G]} > B_{S[B]}$，则第一级应用层重启对象为 $S[E]$。

2）判断下一模块重启群是否属于前面所有重启群并集的子集，是则忽略之，否则将其作为下一级重启对象。本例中 $S[G] \subset S[E]$，忽略该重启群；$S[B] \supset S[E]$，则 $S[B]$为第二级应用层重启对象。

3）应用软件的重启可以释放其所有模块损耗的各种资源，将其作为第三级应用层重启对象 A。

4）操作系统的重新引导可清理所有系统资源，回到初始状态，因此将其作为最后的系统级重启对象 U。

以上分析得到 4 级 TMSRP：$S[E]$，$S[B]$，应用软件 A，整个系统 U。根据每级重启后释放的系统资源和系统的衰退规律，按最大系统可用性原则或最低抗衰成本原则，可以确定各级重启的执行次数 $N[k, i]$，$N[i]$，m，1。其中 $0 \leqslant i \leqslant m$，$0 \leqslant k \leqslant N[i]$，$N[i]$表示第 i 次重启 A 后连续重启 $S[B]$的次数，$N[k, i]$表示第 i 次重启 A，又第 k 次重启 $S[B]$后连续重启 $S[E]$的次数。

2.3　制定 **DMSRP**

制定 DMSRP 的关键是预先设定重启前后的系统性能参数阈值，根据实时监测的参数值决定对哪些程序模块执行重启，以及是否需要执行整个系统重启。衰退规律的稳定程度不同，需要监测的参数也不同。如对衰退规律较稳定的软件，可能只须监测特定的一种或几种系统资源的占用和损耗情况（U_R 和 W_R），据此确定重启时间，对预定模块的重启群执行重启；对衰退规律不稳定的软件，则须监测多个程序模块对特定的一种或几种系统资源的占用和损耗情况（U_R，W_R 和 B_m），并对导致性能衰退的主要程序模块的重启群执行重启；若衰退规律几无规律可循，则必须实时监测所有程序模块对所有系统资源的占用和损耗情况，方能确定执行重启的时间和重启对象。

例如系统有以下衰退规律：物理内存的损耗是导致系统性能衰退的主要因素，损耗物理内存的模块为 E，G，B，损耗程度随机变化，则当检测到

系统性能的衰退时，需分析哪一模块损耗的物理内存最多，将其重启群作为重启对象，实施重启。

3　MSRP 的形式化描述

采用自动机模型，以抗衰粒度和重启次数作为状态划分的依据，描述多级抗衰策略的实施过程。

有限状态自动机 M 为一个 5 元组：$M=\{Q, \Sigma, \theta, q_0, F\}$。其中 Q 为有限状态集，用以记录系统的所有状态；Σ 为输入字母表，表示对系统执行的操作；θ 为转移函数；$Q\times\Sigma\rightarrow Q$，表示通过系统当前状态和执行的操作来确定系统下一状态；q_0 为系统的初始状态；F 表示终止状态集。

3.1　TMSRP 的形式化描述

2.2 节中得到 4 级 TMSRP，为清晰起见，只考虑 2 级 TMSRP，即系统级（U）、应用软件（A）、$S[B]$，其实施过程用图 2 所示的有限状态自动机模型来描述。

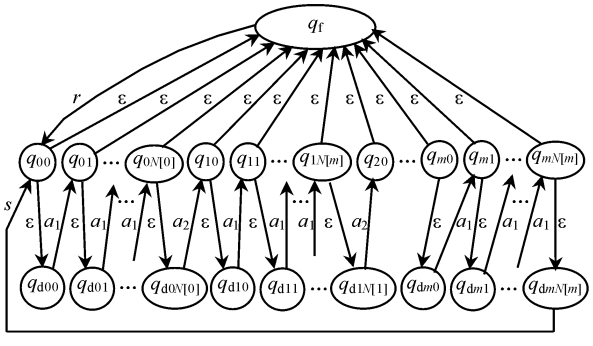

图 **2**　执行三级 **TMSRP** 的系统自动机模型

Fig.2　Automaton model of system using the three levels TMSRP

根据抗衰粒度和重启次数划分系统状态，形成有限状态集 Q，图 2 中以圈表示；带箭头的线表示状态的迁移，其上的标注表示执行的操作，构成自动机的输入字母表 Σ。Q 与 Σ 中元素及其含义见表 1 和表 2。

由转移函数 θ 描述的状态转移规则如下：

$$\theta(q_{i,j}, \varepsilon)=q_{di,j}\quad 0\leqslant i\leqslant m, 0\leqslant j\leqslant N[i],$$

$$\theta(q_{i,j}, \varepsilon)=q_f\quad 0\leqslant i\leqslant m, 0\leqslant j\leqslant N[i],$$

$$\theta(q_{di,j}, a_1)=q_{i,j+1}\quad 0\leqslant i\leqslant m, 0\leqslant j\leqslant N[i],$$

$$\theta(q_{i,j}, a_2)=q_{i+1,0}\quad 0\leqslant i< m, j=N[i],$$

$$\theta(q_{dm,N[m]}, s)=q_{00},$$

$$\theta(q_f, r)=q_{00}。$$

表 **1**　状态集 Q 中元素及其含义

Table 1　The definitions of elements in Q

元素	含义
q_{00}	系统初始状态 q_0/终止状态/重启整个系统 U 后进入的状态/意外失效后重新引导系统进入的状态
q_{d00}	系统第 1 次进入重启前的临界状态（衰退状态）
Q_{01}	系统第 1 次重启 $S[B]$后进入的状态（恢复状态）
q_{d01}	系统第 2 次进入重启前的临界状态
⋮	⋮
$q_{0N[0]}$	系统第 $N[0]$次重启 $S[B]$后进入的状态
$q_{d0N[0]}$	系统第 $N[0]+1$ 次进入重启前的临界状态
q_{10}	系统第 1 次重启应用系统 A 后进入的状态
q_{d10}	系统第 $N[0]+2$ 次进入重启前的临界状态
q_{11}	系统第 $N[0]+1$ 次重启 $S[B]$后进入的状态
q_{d11}	系统第 $N[0]+3$ 次进入重启前的临界状态
⋮	⋮
$q_{1N[1]}$	系统第 $N[0]+N[1]$次重启 $S[B]$后进入的状态
$q_{d1N[1]}$	系统第 $N[0]+N[1]+2$ 次进入重启前的临界状态
q_{20}	系统第 2 次重启 A 后进入的状态
⋮	⋮
q_{m0}	系统第 m 次重启 A 后进入的状态
q_{dm0}	系统第 $\sum_{i=0}^{m-1}N[i]+m+1$ 次进入重启前的临界状态
q_{m1}	系统第 $\sum_{i=0}^{m-1}N[i]+1$ 次重启 $S[B]$ 后进入的状态
q_{dm1}	系统第 $\sum_{i=0}^{m-1}N[i]+m+2$ 次进入重启前的临界状态
⋮	⋮
$q_{mN[m]}$	系统第 $\sum_{i=0}^{m}N[i]$ 次重启 $S[B]$ 后进入的状态
$q_{dmN[m]}$	系统第 $\sum_{i=0}^{m}N[i]+m+1$ 次进入重启前的临界状态
q_f	系统意外失效进入的失效状态

表 **2**　字母表 Σ 中元素及其含义

Table 2　The definitions of elements in Σ

元素	含义
ε	空操作
r	系统意外失效或执行抗衰策略失败后重新引导系统
s	整个系统 U 重启（系统级重启）
a_1	重启 $S[B]$(第一级应用层重启）
a_2	重启应用系统 A（第二级应用层重启）

3.2 DMSRP 的形式化描述

以 2.3 节的 DMSRP 为例，其实施过程如图 3 所示的有限状态自动机模型描述。每次均根据实时监测得到的性能参数来决定应用层重启对象，并检测重启后的系统性能以决定是否需要执行系统级重启。

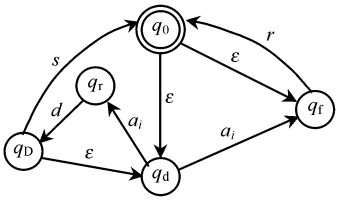

图 **3** 执行 **DMSRP** 的系统自动机模型

Fig.3 Automaton model of system using the DMSRP

该自动机模型的状态集 Q 和字母表 Σ 中元素及其含义如表 3 所示。

表 **3** 状态集 Q 与字母表 Σ 中元素及其含义

Table 3 The definitions of elements in Q and Σ

元素	含义
q_0	初始状态/终止状态
q_d	检测状态二（对重启前性能指标执行检测）
q_r	恢复状态（任一应用级重启后进入的状态）
q_D	检测状态一（对重启后性能指标执行检测）
q_f	系统意外失效进入的失效状态
ε	空操作
r	系统意外失效或执行抗衰策略失败后重新引导系统
S	整个系统 U 重启（系统级重启）
a_i	按实际监测值执行 A，$S[E]$，$S[G]$，$S[B]$ 的重启
d	系统资源损耗程度的检测

由转移函数 θ 描述的状态转移规则如下：

$$\theta(q_0, s) = q_d,\ \theta(q_0, s) = q_f,\ \theta(q_d, a_i) = q_r,$$
$$\theta(q_d, a_i) = q_f,\ \theta(q_r, d) = q_D,\ \theta(q_D, \varepsilon) = q_d,$$
$$\theta(q_D, s) = q_0,\ \theta(q_f, r) = q_0。$$

4 系统停机时间与成本分析

当软件性能衰退时，执行 MSR 可以更大程度地减少停机时间，降低停机成本，但是执行软件抗衰策略同样会引发一定的停机时间和成本，因此必须进行折衷，计算出引发最低成本或提供最高可用性的重启时间间隔。

执行基于时间的抗衰策略时，单级重启最终引发的停机时间和成本由下式计算：

$$t_D = [t_r(1-p(\delta)) + t_R p(\delta)]/T_E,$$
$$C = [t_r C_r(1-p(\delta)) + t_R C_f p(\delta)]/T_E,$$
$$T_E = t_r(1-p(\delta)) + t_R p(\delta)] + \int_0^{\delta}(1-p(t))\mathrm{d}t。$$

执行基于检测的抗衰策略时，相应的计算公式如下：

$$t_D = t_r / T_E,$$
$$C = [C_u T_E + C_r r]/(T_E + r),$$
$$T_E = \int_0^{\infty} t\mathrm{d}(p(t))。$$

以上公式中各变量的意义如表 4 所示。

表 **4** 公式中各变量意义

Table 4 The definitions of variables in the equations

变量	变量意义
t_D	单级重启引发的平均停机时间
C	单级重启引发的平均停机成本
T_E	单级重启的期望周期
t_R	系统意外失效的期望恢复时间
t_r	执行应用级重启的期望时间（系统级重启时替换为 t_{Rr}）
C_f	系统意外失效导致的单位时间恢复成本
C_r	应用级重启的单位时间恢复成本（系统级重启时为 C_s）
C_u	单位时间检测成本
δ	重启时间间隔

5 案例分析

为呈现软件运行过程中的老化现象，分析性能衰退规律，进行了 Web 服务的仿真实验，实验环境包括一台 Web 服务器和一台产生负载的客户机。服务器端使用 Linux 操作系统，安装最新的版本 5.0.5 的 Tomcat，客户端使用开源项目 Jmeter 作为负载生成器。系统资源数据的采集周期为 1 星期，在此期间实验用的机器没有发生宕机。监控的资源包括占用的物理内存（UsedPhysicalMemory）和空闲的交换空间（FreeSwapSpace），采集的时间间隔为 5 min。

对采集到的时间序列经 Harr 小波平滑处理后的结果如图 4 和图 5 所示，其相应的拟合直线分别呈上升和下降的趋势，通过趋势分析可预测系统衰退规律及资源枯竭的时间，结果如表 5 所示。

由表 5 可见，UsedPhysicalMemory 的枯竭时间较 FreeSwapSpace 的枯竭时间短得多，且损耗率也

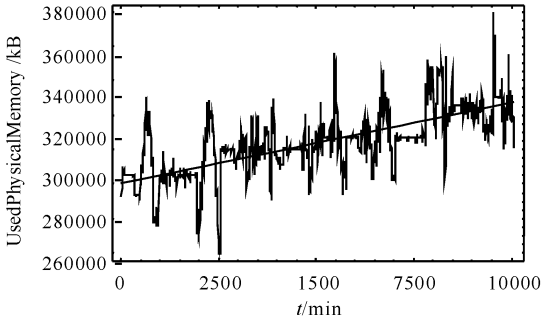

图 **4**　**Haar** 小波平滑过后的物理内存使用序列

Fig.4　The UsedPhysicalMemory sequence smoothed by Haar wavelet

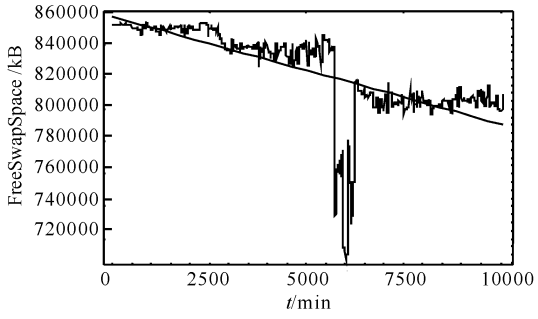

图 **5**　**Haar** 小波平滑过后的空闲交换空间序列

Fig.5　The FreeSwapSpace sequence smoothed by Haar wavelet

大于给定阈值，因此在制定抗衰策略时，应首先考虑物理内存的损耗情况。设定物理内存到达 512 MB 时，系统性能降低到用户无法忍受视为失效状态。图 6 是对系统从健壮状态到失效状态的概率分布函数的统计分析，拟合得到的概率分布函数为

$$p(t) = -0.023\ 37(1-\mathrm{e}^{-0.011\ 176\ 9t}) + 1.023\ 37(1-\mathrm{e}^{-0.000\ 279\ 8t})。$$

其他实验参数如表 6 所示。

表 **5**　系统资源的趋势估计和资源枯竭时间估计

Table 5　The trend and exhausting time estimate of system resources

资源名称	当前值 / MB	占用率阈值 U_L/%	损耗率阈值 W_L/%	警戒值 / MB	趋势估计	资源枯竭时间估计/（5 min）	资源枯竭时损耗率 W_R/%
UsedPhysicalMemory	280.36	100	10	512	0.023 38	9 907.6	45
FreeSwapSpace	857.07	100	10	10.00	−0.033 12	25 575.8	38

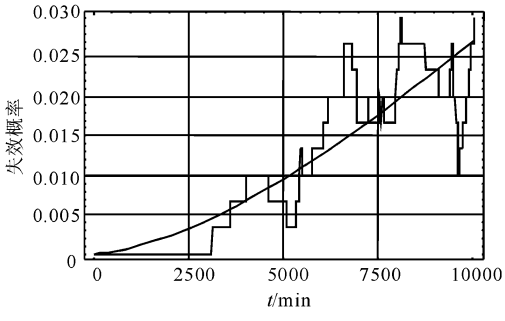

图 **6**　从健壮状态到失效状态的概率分布估计

Fig.6　The probability distribution estimate from robust state to failure state

表 **6**　实验参数列表

Table 6　The list of experimental parameters

项	值	项	值
t_r/min	2	C_r/USD·h^{-1}	500
t_{Rr}/min	6	C_s/USD·h^{-1}	1 000
t_R/h	10	C_f/USD·h^{-1}	5 000

由以上分析可见，系统资源的损耗具有稳定的规律性，应实施 TMSRP；设计的实验较为简单，故将 Web 服务作为抗衰策略的应用层重启对象，执行两级嵌套的软件抗衰策略，最终的系统可用性 A_{En} 和年度期望抗衰成本 C_{En} 由下式计算：

$$A_{En} = 1 - \frac{\sum_{i=0}^{n-1} t_{Di}(\delta_i + t_r) + t_{Dn}(\delta + t_{Rr})}{\sum_{i=0}^{n-1}(\delta_i + t_r) + \delta + t_{Rr}},\ n \geqslant 0;$$

$$C_{En} = 1 - \frac{\sum_{i=0}^{n-1} C_i(\delta_i + t_r) + C_n(\delta + t_{Rr})}{\sum_{i=0}^{n-1}(\delta_i + t_r) + \delta + t_{Rr}},\ n \geqslant 0。$$

图 7、图 8 为系统可用性和年度期望抗衰成本的分析图。图 7 的系统可用性分析显示，当取到某一重启次数 $N=4$ 时，获得最大可用性0.998 086，而执行单一系统级重启的可用性为0.997 931，不执行抗衰策略的可用性为0.997 27。图 8 中抗衰成本的分析显示，当取到某一重启次数 $N=18$ 时，获得最低抗衰成本 $C_E=27\ 969.6$美元/年，而执行单

一系统级重启的成本为44 728.4美元/年，不执行抗衰策略的成本为117 920美元/年。

相应地，可求得执行 MSR 策略的 δ 序列，按最大可用性原则得到 δ＝｛56.863，52.001 5，184.849｝，按最低抗衰成本原则得到 δ＝｛15.453 1，14.936 8，14.486 7，14.089 2，13.734 4，13.414 8，13.124 7，12.859 7，12.616 1，12.391 2，12.182 5，11.988 2，11.806 5，11.636，11.475 8，11.324 6，11.181 6，31.496 2｝，其中最后一个值为系统级重启前的时间间隔。

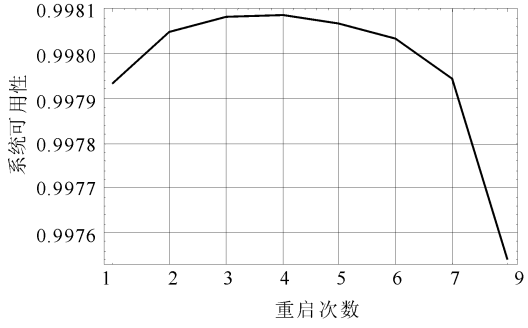

图 **7** 系统可用性分析

Fig.7 The analysis of system availability

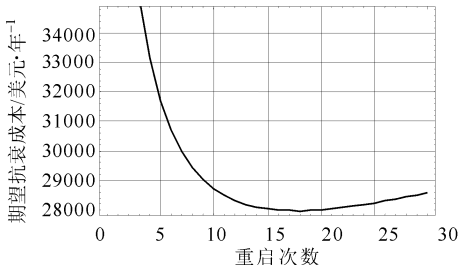

图 **8** 年度期望抗衰成本分析

Fig.8 The analysis of rejuvenation cost per year

综上所述，不同的标准求得不同的 N 值和 δ 序列，因此在实际操作中，需要权衡利弊予以取舍。

6 结论

针对软件老化现象，笔者借鉴 Microreboot 和 RR 的思想，基于 SR 提出了一种细粒度的、事前的、主动的多级软件抗衰技术。通过分析系统资源的占用和损耗情况以及系统体系结构，获知系统的衰退规律，确定抗衰粒度和重启对象，制定相应的抗衰策略，采用有限状态自动机对多级抗衰策略进行形式化描述，通过仿真实验分析策略的制定过程，并给出了实验结果，证明了 MSR 确实可以进一步提高系统可用性，降低抗衰成本。

定义的多级软件抗衰策略不是带检测点或中间状态存储的阶段性重启，而是当系统性能衰退到一定程度时，从当前状态按特定需求（如最低抗衰成本或最大可用性）尽最大努力向初始状态的恢复。这里的尽最大努力是指在当前级别的彻底重启，即该级重启群所占用和损耗的系统资源的完全释放。

理论上讲，MSR 与 SR 可以提供相同的 MTTF，但较之 SR，MSR 可以进一步缩短 MTTR，特别是当系统性能的衰退主要由部分模块对系统资源的消耗引起，而这些模块的重启较之整个系统的重启消耗的时间少得多时，MSR 提供的细粒度重启表现出极大的优势，从而提供更高的系统可用性。

参考文献

[1] Huang Y，Kintala C，Kolettis N. Software rejuvenation：analysis，module and applications［A］. In：Proc of FTCS-25［C］. Pasadena，CA，1995

[2] Garg S，Moorsel A V，Vaidyanathan K. A Methodology for detection and estimation of software aging［A］. In：Proceedings of the 9th International Symposium on Software Reliability Engineering［C］. Paderborn，Germany，1998

[3] Castelli V，Harper R E，Heidelberger P. Proactive management of software Aging［J］. IBM JRD，2001，45(2)：311～332

[4] Vaidyanathan K. Proactive Management of Software Systems：Analysis and Implementation［D］. Department of Electrical and Computer Engineering Duke University，NC，USA，2002

[5] Xiea W，Hongb Y，Trivedi K. Analysis of a two-level software rejuvenation policy［J］. Reliability Engineering and System Safety，2005，87(1)：13～22

[6] Patterson D，Brown A，Broadwell P. Recovery oriented computing（ROC）：motivation，definition，techniques，and case studies［R］. UC Berkeley Computer Science Technical Report UCB/CSD－02－1175，2002

[7] Candea G，Fox A. Recursive restartability：turning the reboot sledgehammer into a scalpel［A］. In：8th Workshop on Hot Topics in Operating Systems［C］. Schloss Elmau，Germany，2001

[8] Candea G，Kawamoto S，Fujiki Y. Microreboot-A technique for cheap recovery［A］. In：6th Symposium on Operating Systems Design and Implementation［C］. San Francisco，CA，2004

[9] Candea G, Cutler J, Fox A. Improving availability with recursive microreboots: a soft-state system case study [J]. Performance Evaluation Journal, 2004, 56(1—3): 213～248

[10] Candea G, Fox A. Crash-only software [A]. In: 9th Workshop on Hot Topics in Operating Systems (HotOS-IX) [C]. Lihue, Hawaii, 2003

Research on Multi-level Software Rejuvenation of Computing System

You Jing[1,2], Xu Jian[2], Li Qianmu[2], Liu Fengyu[2]

(1. *Department of Computer Science & Engineering, Jiangsu Polytechnic University, Changzhou, Jiangsu* 213164, *China*; 2. *Department of Computer, Nanjing University of Science and Technology, Nanjing* 210094, *China*)

[**Abstract**] Recently, the phenomenon of software aging, one in which error conditions actually accrue with time and/or load, has been observed. To counteract software aging, which can cause outages resulting in high costs, a proactive restart technique called software rejuvenation is proposed and the rejuvenation cost is analyzed. In order to reduce the rejuvenation cost and improve software availability and reliability further, rejuvenation granularity should be finer than before. Therefore a fine-grained proactive technique——multilevel software rejuvenation is put forward. Firstly, the degradation law of system performance can be determined by analyzing the occupation and wastage of system resources. Based on the law and the software architecture, the two software rejuvenation policies, i.e. time-based multilevel software rejuvenation policy and detection-based multilevel software rejuvenation policy, can be drafted, and the rejuvenation granularity can be determined. Their formal description of policies is given by finite-state automaton. Finally, the entire process is illustrated with a web service case. This paper provides a case to illustrate the process, and the simulation results of the case show that the multilevel software rejuvenation policy can reduce the MTTR and rejuvenation cost further, comparing with the only system-level software rejuvenation. As a consequence, the system availability and reliability are enhanced.

[**Key words**] software rejuvenation; software aging; system reliability; system availability; automaton