基于访问关系的进程重启相关性判定*)

王 湛 游 静 赵颜利 刘凤玉 张 宏
(南京理工大学计算机系 南京 210094)

摘 要 执行细粒度的进程级软件抗衰,可以进一步降低抗衰成本,提高软件可靠性。本文根据软件系统中进程间控制、调用及数据访问的关系,分析了进程间的耦合度,确定了寻找直接耦合进程的途径,并在此基础上判定了进程重启相关性,从而为实现系统进程级软件抗衰提供了支持。

关键词 软件抗衰,抗衰粒度,重启相关性

Determination of the Restart Dependence Based on the Access Manners of Processes

WANG Zhan YOU Jing ZHAO Yan-Li LIU Feng-Yu ZHANG Hong
(Department of Computer, Nanjing University of Science and Technology, Nanjing 210094)

Abstract In order to save the cost of software rejuvenation and improve software availability and reliability further, the rejuvenation granularity should be improved to the process-level. Based on coupling relation between the processes of the software systems, this paper analyzes the degree of the coupling relation, puts forward a method to search the direct coupled processes, then determines the restart dependence of processes, so it can support to execute software rejuvenation at the process-level.

Keywords Software rejuvenation, Rejuvenation granularity, Restart dependence

1 引言

软件抗衰(Software Rejuvenation)[1]的提出缘于对软件老化现象的认识和对系统可靠性的需求。软件老化是指在软件在长期不间断运行过程中,由于系统内存占用和泄漏、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而导致的软件性能的衰退[1~3]。软件老化最终导致软件失效。而SR是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[2,3]。为了进一步降低抗衰成本,希望将细粒度的抗衰策略执行到进程级[3~6]。

计算系统软件抗衰重启技术研究中提出的重启树(restart tree)[10]和George Candea提出的微重启(Microreboot)[8,9]都是从模块执行恢复,没有从更细粒度的进程级执行恢复。要执行进程级的SR,可以借鉴文[8,9]的思想,首先查询系统信息,确定进程耦合度,制定寻找直接耦合进程的方法,从而最终得到进程重启相关性。

本文根据系统提供的各种信息,给出了判定进程间的耦合程度的标准,确定了寻找直接耦合进程的途径,并在此基础上得到进程重启相关度,最终确定进程重启相关性,为实现系统进程级抗衰提供支持。

2 进程与进程耦合性

软件包括计算机程序、程序所使用的数据及有关的文档资料。软件分为系统软件和为特定应用建立的应用软件。系统软件与计算机硬件交互,是处理不确定数据结构的程序,即操作系统。它们是底层通用的软件,为特定应用建立的应用软件往往是建立在它们之上的。不同的软件系统皆由协同工作的软件进程构成[12]。进程就是一个正在执行的程序,包括程序计数器、寄存器和变量的当前值[11]。

尽管每个进程是一个独立的实体,但进程间经常发生交互作用,这种进程连接的紧密程度称为耦合性。进程间的连接越紧密,联系越多,耦合程度越高。根据进程间交换数据的情况,通常将耦合性分为5类:非直接耦合、数据耦合、控制耦合、公共耦合和内容耦合。

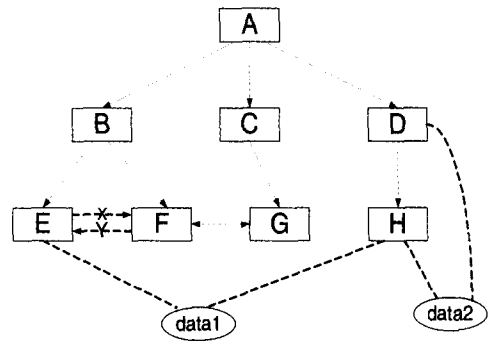

图1 直接耦合进程关系结构图

如果进程间的联系只是通过上层进程的控制和调用来实现,称为非直接耦合。若上层进程调用下层进程,完成指定功能,且调用关系是通过参数传递来实现的,称为控制耦合。若进程直接进入另一进程中存取数据或使用服务,或存在双向调用关系,称为内容耦合。图1中,进程H与E、D与H分别通过一个公共环境进行数据交换,称为公共耦合。这个公共环境可以是全局变量、全局数据结构、通讯缓冲区和数据(库)

*)国家自然科学基金项目(60273035)、国防科工委基础应用项目(K1704060511)。**王 湛** 博士研究生,主要研究方向:软件性能保持;**游 静** 博士研究生,主要研究方向:软件性能保持;**赵颜利** 博士研究生,主要研究方向:计算机视觉、图形与图像;**刘凤玉** 教授,博导,主要研究领域:人工智能与信息安全;**张 宏** 教授,博导,主要研究领域:人工智能与信息安全。

文件等。如果进程间通过接口的参数表交换信息数据，称为数据耦合，如图1中进程F和E。上述5种耦合的耦合度依次增加，其中内容耦合的适性最差。本文将采取直接交互方式而发生数据、控制、公共、内容4种耦合合关系的进程称为直接耦合进程。

3 直接耦合进程的判定

系统软件是底层通用软件，而应用软件是建立在它们之上的为特定应用而建立的，因此确定不同软件系统中进程间耦合性及查找相关进程的方式和途径也就有所不同。

3.1 系统软件的直接耦合进程的判定

在系统软件中各组成部分的运行情况都是可查获的，故而进程间的耦合性是可以通过查询信息的途径获取的。在此系统中，每个进程都占用一个进程表项，包含进程的各种信息。通常，进程表中存有进程管理、内存管理和文件管理的有关信息[11]。

进程间最简单的关系是创建与被创建。当一个进程被创建时，父进程得到子进程的进程标志符，由此可查找到子进程，进而可查找到更深层次的子进程的树形数据结构。同样也可以通过查找子进程的进程表项中内存管理信息中的父进程项，找出其父进程。子进程被创建后，父子进程将完全独立发展，同时父子进程共享打开的文件。此时可以确定父子进程间的一项耦合关系——控制耦合。

一个组是进程的集合，它们按照某种系统或用户指定的方式一起工作[11]。系统分组的方式有两种：层次分组和对等分组。对等的分组是对称的，没有哪一个进程是关键。若其中一个进程崩溃了，组内其他进程不会受到什么影响。层次分组正好相反。协调进程的崩溃会使得整个组不能工作。系统进程表中内存管理信息中的组标志符及进程组项可以确定该进程的其它组员。若该系统采用对等分组，则此进程与其他组员间属数据耦合；若系统采用层次分组，且此进程是协调进程，则该进程与其他组员间属控制耦合；如果该进程并非协调进程，则该进程与协调进程存在控制耦合，而与其它组员间属非直接耦合。

进程除了通过上两种简单的方式发生交互作用外，还可以通过其它通信的方式实现其耦合关系，其通信的方法包括信号量、管程、消息传递、软中断。

·进程通信采用信号量、管程方式时，进程间共享内存。每个进程的地址空间由代码段、数据段、堆栈段组成。堆栈段和数据段是不能共享的，只能共享代码段。系统进程表内存管理信息中记录了代码段指针。如果代码段被共享，代码指针指向共享代码表，从而可以得出与此进程共享内存的直接耦合进程。若系统中包含分页机制的话，也可由类似的方法查获。此时相关进程属数据耦合。

·消息传递通信方式在不同的系统中实现的方法不同。通常有管道和信箱两种方法。在有些系统中，两进程间可以生成一种通道，其中一个进程可以在通道中写入字节流而让另一个进程读取，这种通道叫做管道。如UNIX系统中，当shell得到命令sort<f | head，此时进程sort与head之间建立了管道。此时两进程间传送数据不需使用文件，而是从管道中直接存取。此时进程间属控制耦合。有时进程也会以ASCII文件作为输入和输出，此时该文件与管道的作用及执行方式是一样的。在分布式系统中，通常采用信箱的方法实现进程间的消息传递。可以通过查询进程表进程管理信息中的消息队列指针，得到其消息队列，进而查出与之消息传递的直接耦合进程。此时进程间属公共耦合。

·紧急状况时，进程间也可以通过软中断的方式进行通信。一个进程可以给另一个进程发信号，通过信号决定忽略、捕获还是删除另一进程。此时可通过查询进程表的信号项中进程接收、发送信号信息，得出其直接耦合进程。此时进程间属控制耦合。

由于文件在系统软件中的重要作用和广泛用途，进程间的交互作用经常通过文件系统实现。

进程间经常需要共享文件或用特殊文件表示的I/O设备。当一个进程运行时，首先查询其进程表中的用户标志符和组标志符，进而查询共享资源的存储控制表来确定该进程访问的文件和I/O设备。若该资源存在其它用户，则进一步查询该共享资源的访问控制表，从该表中可以获得其所有用户、用户组及其允许的访问方式(读、写、执行)信息，进而对照进程表，获得该进程的直接耦合进程。两进程间对共享资源不同访问方式的组合与其耦合性对应关系由表1给出。

表1 进程不同访问方式的耦合性

耦合性 访问二 / 访问一	R	W	X	RW	RX	WX	RWX
R	1	2	2	2	2	2	2
W	2	1	4	2	4	4	4
X	2	4	1	4	2	4	4
RW	2	2	4	2	4	4	4
RX	2	4	2	4	2	4	4
WX	2	4	4	4	4	4	4
RWX	2	4	4	4	4	4	4

表中，1为数据耦合，2为控制耦合，3为公共耦合，4为内容耦合。

3.2 应用软件直接耦合进程的判定

在应用软件系统中，进程的体系结构给出了进程是如何构造的。此时每个机器的进程称为结点。当一个系统是由多个结点上的进程组成时，进程体系结构设计必须指明每个独立进程的功能、在哪个结点上运行以及进程是如何通信的。进程间主要侧重基于嵌套字的消息传送通信。进程通信的工具是状态机。在UML中，状态机可以模拟待开发系统的动态行为。通过查询状态机，可确定其相关状态及其耦合关系，状态机中的状态即是进程[12]。图2、图3所示为加入网络游戏的状态机。

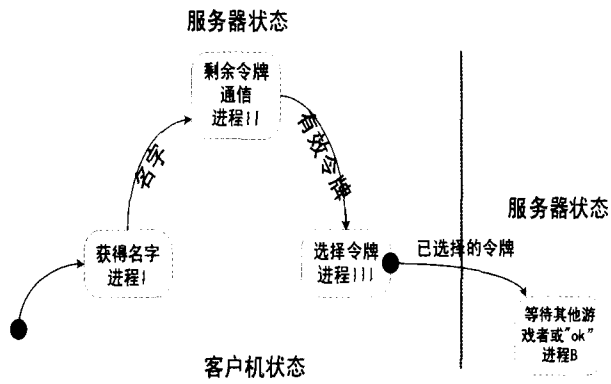

图2 进程A的子状态机

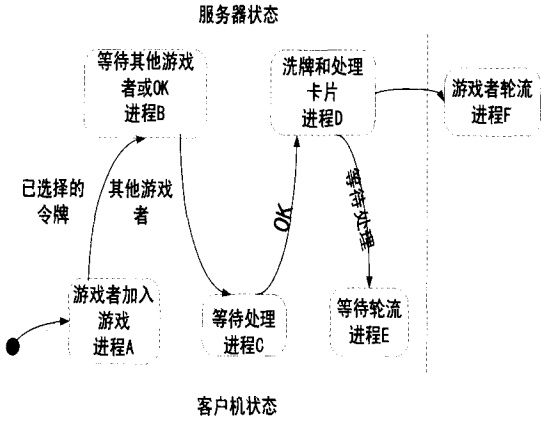

图3 游戏者加入游戏的状态机

该系统的任务是协调互联网上远程游戏的进行。游戏者输入用户名,以此得到一个令牌,然后进入等待其他游戏者或OK的状态;此时一旦至少有3个游戏者加入,而最先加入者选择了"OK"按钮,则游戏开始,进入"洗牌和处理卡片"子状态,该子状态在每个游戏者之间进行通信并无条件进入"游戏者轮流"。由此可得出各个进程间如下的直接耦合关系。

控制耦合:

进程I 与 进程A 进程II 与 进程A

进程III 与 进程A 进程I 与 进程II

进程III 与 进程B 进程A 与 进程B

进程C 与 进程D 进程D 与 进程F

数据耦合:

进程I 与 进程II

图3中进程D到F并未标志任何触发事件,但进程D的失败将导致进程F的失败,故属控制耦合。

4 重启相关性的判定

当系统需要重启一进程时,其它有些进程需要同时重启,则将该进程及其所有重启进程视为一个重启群。在对特定进程执行重启操作前,必须先计算其重启相关度和判定其重启相关性,否则会出现数据丢失、数据不一致,甚至软件失效等运行错误。

4.1 重启相关性

一个进程重启时,可能导致其它进程出现故障或错误,称为进程间的重启相关性。进程间重启相关性取决于它们的耦合程度。一个进程重启时,与其耦合程度越低的进程出错的可能性也越低。对应于进程间的几种耦合性,将重启相关性分为4类:相互独立、功能相关、状态相关和功能双向相关。定义如下。

定义1 若进程A调用进程B,而进程B未调用进程A,则A与B功能相关,记为A→B。

性质1 功能相关具有传递性:若A→B,B→C,则A→C。

定义2 若进程A调用进程B,且进程B调用进程A,则A与B功能双向相关,记为A↔B。

性质2 功能双向相关具有交换性和传递性。若A↔B,则B↔A;若A↔B,B↔C,则A↔C。

定理1 若进程A与B功能相关,则A重启,B同时重启。但B重启时,A不一定重启。若进程A与B功能双向相关,则其中一个进程重启,另一进程同时重启,即A与B总是同时重启。

由以上的定义和性质可知,图1中,A→B,A→C,A→D,B→E,B→F,C→G,D→H,F↔G。由性质1知,A→E,A→F,A→G,A→H。此时,当进程B重启时,进程E同时重启。如若不然,被调用进程E的状态不能与调用进程B的状态保持一致。但E重启时,B不一定重启,因为E并不向其上层进程传递控制和调用参数,不会影响其状态。当进程F重启时,由上可知,进程G一定同时重启,反之亦然。

定义3 若进程A与B有数据或状态共享,则称A与B状态相关,记A∧B。

性质3 状态相关具有交换性和传递性:若A∧B,则B∧A;若A∧B,B∧C,则A∧C。

定理2 若进程A与B状态相关,则其中一个进程重启,另一进程同时重启,即A与B总是同时重启。

由图1知进程D与H、E与H状态相关,即D∧H,E∧H;由状态相关的传递性知D∧E,此时D与E必同时重启。因为D间接使用了数据1中的数据,不同时重启,可能出现数据不一致或数据冲突。

定义4 若进程A与B既非功能相关也非状态相关,则A与B相互独立,记为A∅B。

性质4 相互独立具有交换性:若A∅B,则B∅A。

在数据耦合和非直接耦合的情况下,相应的进程认为相互独立。相对独立不存在传递性。

4.2 重启相关度与相关性确定

当两进程非直接相连时,判定它们的重启相关性需要考虑其间经过的所有进程间的重启相关性。

定义5 一个进程A重启与另一进程B重启的相关程度称为重启相关度D[A·B],取值为{0,1,-1,2,3,4}。具体地,D[A∅B]=0,D[A→B]=1,D[A∧B]=2,D[A↔B]=3。另外,定义D[A·A]为进程A自身重启相关度,且D[A·A]=4。特别地,若A→B,则记B与A的重启相关度为D[B·A]=-1。

由上述,根据重启相关类型可以确定重启相关度。同样地,由重启相关度可以确定重启相关类型。即D[A·B]=0/1/2/3⇔A相对B独立/A与B功能相关/A与B状态相关/A与B功能双向相关。

约定:当两个进程之间存在多种重启相关性时,取重启相关度最高的相关性。当两个进程之间存在多种耦合时,取最高的耦合。耦合程度越高,进程间连接关系越紧密,重启相关度也越高。这样可以保证重启的完善性和有效性。

定义6 若进程A与B功能相关,认为从进程A可达进程B,但从进程B不可达进程A;若进程A与B状态相关或功能双向相关,认为进程A与B相互可达;若进程A与B相互独立,认为进程A与B相互不可达。反之亦然。

定义7 若进程A与K非直接连接,但A经若干进程可达K,则认为从进程A到进程K间有一条可达路径R[A~K],经历的进程以"-"连接。

定理3 若从进程A到K存在可达路径R[A~K]=A-K1-K2-…-Kn-K,则进程A与K的重启相关度为D[A·K]=Min{D[A·Kn],D[Kn·K]}。

因为两个不直接连接的进程间的耦合程度取其路径内最松散的耦合,而耦合程度越低,重启相关度越小。由定理3,图1中进程A与H的重启相关度为D[A·H]=Min{D[A·D],D[D·H]}=Min{D[A→D],D[D∧H]}=Min{1,2}=

1，再由定义5，可知进程A与H功能相关A→H。

推论1 若进程A与K间存在可达路径R[A～K]，但不存在可达路径R[K～A]，则进程A与K功能相关。

推论2 若进程A与K间既不存在可达路径R[A～K]，也不存在可达路径R[K～A]，则进程A与K相互独立。

推论3 若进程A与K间既存在可达路径R[A～K]，又存在可达路径R[K～A]，则进程A与K或状态相关或功能双向相关。

对推论1进行证明，用反证法。若进程A与K非功能相关，则可能是其它3种情况：相互独立、状态相关、功能双向相关。若进程A与K相互独立，按定义6，A与K相互不可达，即不存在可达路径R[A～K]，与假设矛盾。若进程A与K状态相关或功能双向相关，由定义6，进程A与K相互可达，再由定义7，两进程间必存在一条可达路径R[K～A]，与假设矛盾。因此进程A与K必为功能相关。对于推论2，3，可用类似方法推出。

由以上分析，要确定各进程间的重启相关性，需以下步骤：

①确认初始进程和终结进程，因为可达路径是单向的。

②查找从初始进程到终结进程的所有可达路径。

③按每一条可达路径求初始进程和终结进程间的重启相关度。

④确定初始进程和终结进程的最终重启相关度及重启相关性。

按以上步骤可得图1所示的进程关系结构图，对应的各进程间的重启相关度如表2所示。

表2 图1中任意进程间重启相关度

重启相关度 初始进程＼终结进程	A	B	C	D	E	F	G	H
A	4	1	1	1	1	1	1	1
B	−1	4	0	1	1	1	1	1
C	−1	0	4	0	0	1	1	0
D	−1	−1	0	4	2	0	0	2
E	−1	−1	0	2	4	0	0	2
F	−1	−1	−1	0	0	4	3	0
G	−1	−1	−1	0	0	3	4	0
H	−1	−1	0	2	2	0	0	4

当具体实施该策略时，首先确定各影响因素的损耗阈值。当其到达该阈值时，查找问题进程，确定其直接耦合进程与耦合性，在此基础上判定进程间的重启相关性，进而得到进程重启群。之后开始执行重启，实现进程级软件抗衰，防止软件失效。

结论 要执行进程级的软件抗衰，进一步降低抗衰成本和提高软件可靠性，需具备3个前提条件：一是系统中各部分运行情况可监控；二是进程间耦合关系及直接耦合进程可确定；三是在二的基础上查找非直接耦合进程间的路径，确定该进程与其它进程间的重启相关性，建立合理的进程重启群。一般的软件系统都满足第一个条件，问题的关键就在于查找直接耦合进程。

本文根据通常的软件系统提供的信息及进程间控制、调用及数据访问关系，给出了进程耦合关系的分类，分析了进程间的耦合程度，确定了查找直接耦合进程的途径，并给出了进程重启相关性及相关度的判定方法。据此可确定进程重启群，实现了进程级软件抗衰。从而更大程度上为实现细粒度的软件抗衰提供了支持，更有效地节约了抗衰成本，提高了软件的可靠性。

参考文献

1 Garg S, Moorsel A V, Vaidyanathan K. A Methodology for Detection and Estimation of Software Aging. Proceedings of the 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998

2 Huang Y, Kintala C, Kolettis N. Software Rejuvenation: Analysis, Module and Applications. In: Proc. of FTCS-25, Pasadena, CA, 1995

3 Castelli V, Harper R E, Heidelberger P. Proactive Management of Software Aging. IBM JRD, 2001, 45(2): 311～332

4 Hong Y, Chen D, Li L. Closed Loop Design for Software Rejuvenation. Workshop on Self-Healing, Adaptive and self-MANaged Systems (SHAMAN), New York, 2002

5 Xiea W, Hongb Y, Trivedi K. Analysis of a two-level software rejuvenation policy. 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. [Technical Report]. UC Berkeley Computer Science UCB/CSD-02-1175. 2002

7 Candea G, Fox A. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalpel. 8th Workshop on Hot Topics in Operating Systems, Schloss Elmau, Germany, 2001

8 Candea G, Kawamoto S, Fujiki Y. Microreboot - A Technique for Cheap Recovery. 6th Symposium on Operating Systems Design and Implementation, San Francisco, CA, 2004

9 Candea G, Cutler J, Fox A. Improving Availability with Recursive Microreboots: A Soft-State System Case Study. Performance Evaluation Journal, 2004, 56(1-3): 213～248

10 游静，徐建，张琨. 计算系统软件抗衰重启技术研究[J]. 信息与控制(已录用，待发)

11 Tanenbaum A S. Modern Operating Systems[M]. 北京：机械工业出版社，1999

12 王庆育. 软件工程[M]. 北京：清华大学出版社，2004

(上接第252页)

法NMR脉冲序列的设计方法基本上是合理的。同时，尽管一般化量子搜索算法对经典Grover算法进行了有效的扩展，但在具体应用时，经典Grover算法的实现开销还是最经济的。量子算法在传统计算机上进行的NMR模拟实现，对量子计算理论研究，量子算法的理论、可行性和正确性研究，以及量子计算机的物理实现都具有一定的参考价值。

参考文献

1 Grover L K. A Fast Quantum Mechanical Algorithm for Database Search [C]. In: Proc. of the 28th Annual ACM Symposium on Theory of Computing, 1996. 212～219

2 Grover L K. Quantum computers can search rapidly by using almost any transformation [J]. Phys Rev Lett A 80, 1998. 4329～4332

3 Miao X. Universal Construction of Unitary Transformation of Quantum Computation with One- and Two-body Interactions [J]. http://xxx. lanl. gov/abs/quant-ph/0003068

4 Michielsen K, Raedt H D. QCE: A Simulator for Quantum Computer Hardware [J]. Turk J Phys, 2003, 27: 129

5 http://rugth30. phys. rug. nl/compphys

6 Long G L, Li Y S, Zhang W L, et al. Phase matching in quantum searching [J]. Phys Lett A, 1999, A262: 27～34

7 Chuang I L, Gershenfeld N A, et al. Bulk quantum computation with nuclear magnetic resonance: theory and experiment [J]. Proc R Soc Load A, 1998, 454: 447～467

8 Long G L, Yan H Y, Li Y S, et al. Experimental NMR realization of a generalized quantum search algorithm [J]. Phys Lett A, 2001, A286: 121～126