多进程系统状态恢复的一种优化方法*

王 准[1,2] 陈俊亮[2]

[1](南方通信(惠州)实业有限公司北京研究所 北京 100086)

[2](北京邮电大学程控交换技术与通信网国家重点实验室 北京 100876)

E-mail: wangzhun@263.net

摘要 在由多进程组成的分布式系统中,为了保证系统在失效之后进程状态恢复的一致性,某一进程的状态回卷将引起所有与之有依赖关系的进程的后退.已有的依赖关系定义过于严格,该文重新定义了进程状态之间的依赖关系,可以减小进程状态卷回所造成的影响.

关键词 软件容错,状态一致性恢复,恢复块.

中图法分类号 TP316

软件容错的冗余策略基本上可以分为空间冗余和时间冗余两大类.其中时间冗余主要在B. Randell[1]等人提出的恢复块(recovery block)的基础上展开.所谓恢复块,最初是针对顺序进程而提出的,是指在进程的执行过程中或者在某些指定的执行点对进程的状态做接收测试,如果有错误存在,则将进程恢复到某一无错的执行状态,并试图通过采用同种方法或不同种方法再试的手段得到正确结果.在恢复块方法应用于多进程的并发系统时,如何进行快速有效的系统状态恢复显得十分重要.

一般说来,进程的状态恢复可以分为协作的(coordinated)和非协作的(non-coordinated)两种恢复方法.协作的恢复方法是,将系统在动态执行过程中进程之间的消息交互划分为互不相交的若干单元,每个单元均包含了两个或更多的进程及其在特定时间段内的消息交互过程,这些单元称为"会话"(conversation)[1].会话的执行具有封闭性,组成会话的各个进程在会话过程中不允许与非会话进程交互;会话的原子性则要求会话中任何进程的错误均导致所有参与进程卷回至进入会话前的状态.由于系统的复杂性降低为会话的复杂性,通过对会话的精心设计,进程的卷回距离得到有效的控制,还可以防止多米诺现象的发生.针对不同的应用领域,会话方案衍生出各种变形,如Exchange[2],目前也已存在支持会话的程序设计语言.采用协作的恢复方法,必须由进程的设计人员指定进程需要保存的状态(称为备查点),备查点的设置对应用程序不透明,但备查点的备查间隔和数目可以根据具体应用优化,从而降低开销.

非协作的恢复方法对于应用程序透明,由系统的容错机制自动为各个进程设置备查点以备恢复之用.非协作的恢复方法在大型的计算型应用中被普遍采用,一般采用定期保存备查点的方案,备查间隔在几十分钟至数小时不等.在有大量进程交互的系统中采用该方法尚处于研究阶段,一般的想法是以Lamport[3]提出的"先于(happened before)"关系为基础,确定各个进程状态之间的依赖关系(或者因果关系(causual relation)[4]),在某一进程作状态恢复时,所有对其具有依赖关系的进程均需做适当的状态回卷以保证全局状态的一致性[5].该方法的缺点在于,为了防止多米诺现象的发生,需要设置的备查点过多[6],系统的执行效率因此要大打折扣.所以,很多人主张不采用单一的备查点方法,而与消息日志法结合起来共同使用.

在协作的恢复方法中,会话的封闭性要求会话的参与进程在会话的过程中不得与任何非会话进程交互.因此,如果会话之间存在着共同进程的话,只能以串行的方式执行这些会话.如果具有这种特点的会话大量存在,

---

* 本文研究得到国家教委博士点基金资助.作者王准,1971年生,博士,主要研究领域为软件容错,通信软件,分布式系统.陈俊亮,1933年生,教授,博士生导师,中国科学院院士,中国工程院院士,主要研究领域为程控交换系统,智能网,通信软件.

本文通讯联系人,王准,北京 100086,北京市9628信箱41分箱0901室

本文1997-12-05收到原稿,1998-04-17收到修改稿

并且会话的执行时间较长,对系统执行效率的影响是比较大的.

在非协作的恢复方法中,由于遵循依赖性后退法则,一个进程的后退势必引起所有对其具有依赖关系的进程的后退.如果大量的进程都对某一进程具有依赖性,则该进程的后退导致的后退进程数目将可能十分庞大.因此需要较长的系统状态恢复时间.

我们希望消息交互的并发性,处于同一依赖关系树中的进程数目越少越好.这样,进程之间更多的是并发关系,从而不论是采用协作的恢复方法,还是采用非协作的恢复方法,单一进程失效后系统的恢复面都可以做到很小,同时也有利于系统在无错时的运行效率.

本文提出一个优化算法,通过重新定义进程状态的依赖关系,减小进程状态回卷的影响面.有关进程的状态恢复可以不必通过卷回的手段实现.在满足系统功能性需求的前提下,允许系统在一定的时间段内存在传统意义上进程之间状态不一致的现象.采用本方法的优点是:在采用协作的恢复方法时,可以将有关进程从会话中脱离开来,从而增加了会话之间并发执行的机会;在采用非协作的恢复方法时,可以减少单一进程失效后系统的恢复面.

本文第1节提出算法成立的若干假设.第2节陈述算法内容,第3节是与前人的工作做一些比较.

1　假　设

我们的算法建立在以下假设模型之上:

(1) 系统是由 N 个进程构成的集合.进程之间以消息发送/接收的方式进行通信,除此之外,不存在任何其他进程之间交互的方式.系统的状态是系统内各个进程状态的集合.由于进程之间通信的异步性和并发性的存在,同时出于实现复杂性的考虑,不要求这些进程的状态必须同时是系统在某一时刻相应进程的状态.

(2) 通信的可靠性是保证的.不会出现消息出错、丢失或重复接收的情况.

(3) 进程的类型为激励-响应方式.即进程在大部分时间内处于等待消息的状态,进程的下一步动作取决于收到消息的内容.进程在处理完消息之后,又重新回到等待消息的状态.

(4) 进程的失效语义[7]是失效后停止(fail-stop),并且不考虑同一时刻两个或更多进程失效的情况.

我们首先援引 Lamport 关于"先于(happened before)"关系的定义[3].

定义 1.1. 在分布式系统中,事件可能是进程自身状态的改变,也可能是接收或者发送消息.事件 a 先于事件 b 发生,当且仅当:

(1) a 和 b 是同一进程内的事件,并且 a 在 b 之前发生;或者

(2) a 是某进程发送消息 m 的事件,b 是另一进程接收 m 的事件.

(3) 先于关系可用符号"$<$"表示,"$<$"具有传递性,即 $(a<b)\wedge(b<c)\Rightarrow a<c$.

以往的文献一般将进程状态之间的依赖关系等价于先于关系,即 $a<b\Leftrightarrow R_a\rightarrow R_b$,其中符号"$\rightarrow$"表示依赖关系,$R_a$ 表示事件 a 发生后进程的状态.Leong 等人[8]认为这种依赖关系过于严格,并引入了消息语义的概念,将对其他进程状态不造成影响的消息定义为非重要消息(insignificant message),非重要消息的传递不构成依赖关系,从而减小了进程状态依赖树的规模.非重要消息的定义如下.

定义 1.2. 由进程 P_i 发往进程 P_j 的消息 m 是非重要的,只要下述条件之一成立:

(1) m 没有引起 P_j 显式或隐式的状态变化(所谓显式的状态变化,是指进程的内容发生了改变.隐式的状态变化是指进程有消息输出).

(2) m 没有引起 P_j 状态的显式变化,并且 m 仅产生非重要的消息.

(3) m 引起了 P_j 状态的显式变化,但是该变化被 P_j 接收的下一条消息 m' 所重写,并且 m 仅产生非重要的消息.

(4) m 引起了 P_j 状态的显式变化,但是该变化被 P_j 未来接收的某条消息 m' 所重写;并且 m 仅产生非重要的消息,所有在 m 和 m' 之间提交给 P_j 的消息均是非重要的.

如果 m 不是非重要消息,则它是重要消息.

进程之间的依赖关系可以通过重要消息导致的"先于"关系来确定.但我们认为该消息语义仍有进一步扩充

的可能性.

系统在某个进程失效后,首先将失效进程卷回至最近的一个无错状态,然后寻找包含该状态在内的一个全局一致性进程状态.进程的状态是一致的,是指不存在这样一种情况,即重要消息在某一进程的保存状态之后发送,在另一进程的保存状态之前接收到.进程状态不一致的直接现象是产生了孤儿进程.

定义 1.3. 设 m 是由进程 P_i 发送至 P_j 的一条重要消息.如果 P_i 保存了发送 m 之前的状态 R_i,P_j 保存了接收 m 之后的状态 R_j,则状态 R_i 和 R_j 不一致,并且将重要消息 m 称为孤儿消息,将处于状态 R_j 的进程 P_j 称为消息 m 引起的孤儿进程.

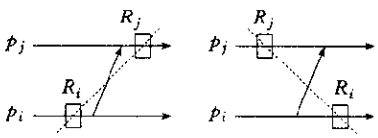

图1

图1给出了不一致状态和一致状态的两个例子.

在恢复进程状态时,如果进程的重演是确定性的,即可以保证 P_i 在重演的过程中向 P_j 重复发送了消息 m,则认为孤儿消息在恢复的过程中被消除了,P_j 亦不再是孤儿进程.这是乐观的消息日志方法解决进程状态不一致问题的手段之一[9],并且要求消息的接收进程有容忍重复接收消息的能力.但如果进程的重演是不确定的(即不能保证 P_i 向 P_j 发送了同样内容的消息 m,或者虽然发送了相同内容的消息,但是顺序同以往不同),则 P_j 将永远成为孤儿进程.

通过对应用进程的考察,我们有两个发现.第1,在某些应用中,对于某些消息引起的孤儿进程,即使不作状态的卷回也不会影响系统功能的正确性,仅可能影响到系统的性能(在后面的章节中,我们将举出相应的例子).此类进程往往与其他进程交互频繁,因此,单独对这种进程进行分析和优化可能有一定的意义.第2,某些进程的关键状态(根据具体应用不同而不同)可以根据其他进程的状态进行重建,只要系统的进程之间存在着数据的冗余(注意,这种冗余是出于非容错的原因,如效率等设计的),就存在着进行一定程度进程状态重建的可能性.

在第2节,我们将在这两个发现的基础上,给出进程状态恢复的优化策略.

2 优化策略

下面,我们来考察交换软件中的资源管理进程.

对于交换系统中的每一类资源,如用户线、中继线和交换通路等,都有相应的资源管理进程负责管理.一般来讲,在呼叫的建立阶段,呼叫进程以一定的顺序逐一向相应的资源管理进程申请资源,在得到满足之后建立连接,在呼叫的释放阶段再以一定的次序释放这些资源.这决定了资源管理进程几乎和所有的呼叫进程交互.以话音交换通路管理进程(sp)为例,典型的交互过程如图2所示.其中 m_1 和 m_3 分别是进程 P_1 和 P_2 申请交换通路的消息,m_2 和 m_4 是相应的应答消息.管理进程在接收到申请消息(m_1 和 m_3)后,首先寻找是否存在空闲的话音交换通路,如果存在空闲通路,则选定其中的一条,并将该通路置忙之后向申请进程返回该通路的编号(m_2);如果不存在空闲通路,则向申请进程返回失败消息(m_4).m_n 和 m_{n+1} 是释放话路的过程,其中 m_n 携带有释放话路的编号,话路管理进程将该话路置闲之后返回一个应答消息(m_{n+1}).随后,该话路即可被其他进程所占用.

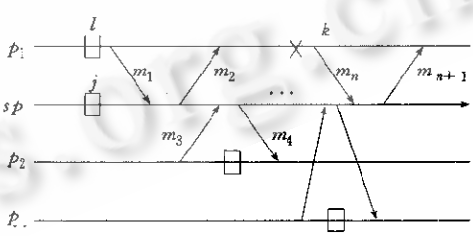

图2

我们现在考虑进程 P_1 在 k 点失效的情况.假设 P_1 在失效后状态退回到某个呼叫稳态 l,根据定义1.2所导致的依赖关系,sp 的状态必须后退至接收消息 m_1 之前(这里是 j),从而引起 P_2 以及其他所有对 j 状态有依赖关系的进程的后退.由于 sp 在接收到 m_1 之后可能与大量的进程进行了交互,因此,需要后退的进程可能很多.在实际应用中,有的后退是不可能的,如,进程向外界环境输出的消息不可能收回;有的后退是不明智的,如,后退影响了正在进行的通话.

事实上,不必进行 sp 的状态后退,只需将 P_1 卷回至状态 l,因为 P_2 等进程并没有受到 P_1 失效的影响.P_1 失效给 sp 带来的负面效应仅仅是某一条话路的虚占.这可以通过其他方法(可能的话,可与硬件话音通路寄存器

核对[10])对 sp 的状态进行修正.

下面,我们提出对于该类进程的一般性的状态恢复算法.

首先,我们考虑像交换系统这样的多用户系统的功能相对正确性.

定义 2.1. 进程可以用有限状态自动机 $F=(Q,E,\delta,q_0)$来表示,其中 Q 是进程状态的有限集合,E 是事件的有限集合,δ 是一个状态转换函数 $\delta:Q\times E\to Q$,q_0 为进程的初始状态.

假设进程 p 和 p',我们用 $p\parallel p'$表示一个进程,其行为就是 p 和 p'构成的系统的行为.符号"$\parallel$"称为并发算子.我们将系统中进程的集合记为 S.环境可以看做是另外一个多进程的系统,其中的每一个进程表示一个接受服务的用户,称为环境进程,环境进程的集合记为 S^{ENV}.

引入迹的概念和相应的算子:

迹:一个进程行为的迹是事件的一个有限序列.

$\circ$:连接算子($a\circ b$)将两个迹 a 和 b 按次序简单地拼到一起,构成一个迹.

$\uparrow$:局限算子($t\uparrow A$)表示迹 t 局限于集合 A 中的事件,即把 t 中所有不属于 A 的事件去掉之后留下的迹.例如:$\langle e_1,e_2,e_4,e_3,e_4\rangle\uparrow\{e_1,e_4\}=\langle e_1,e_4,e_4\rangle$.

$\leqslant$:前缀算子.假设 t'是序列 t 的一个初始子序列,则我们总能找到 t'的某个扩展序列,使 $t'\circ\mu=t$.定义 $t'\leqslant t\equiv(\exists\ \mu\cdot t'\circ\mu=t)$.

进程 p 的所有可能的迹的全集是有可能事先知道的,我们定义函数 $Trace(p)$来产生这个集合.

定义 2.2. 进程 p 的功能对于进程 p'是相对正确的,当且仅当

$$\forall\ t\in Trace(p')\cdot(\exists\ t'\in Trace(p)\cdot(t\uparrow Comm(p,p')=t'\uparrow Comm(p,p'))),$$

其中 $Comm(p,p')$表示 p 和 p'交互事件的集合.

如果 $p=\parallel_{p_i\in S}p_i$,$p'\in S^{ENV}$,则称系统对于用户 p'的功能是相对正确的.

定义 2.3. 系统的功能是相对正确的,当且仅当

$$\forall\ p'\in S^{ENV}\cdot\text{系统对于用户 }p'\text{的功能相对正确.}$$

定义系统功能的相对正确性,仅从单个用户的视角来看待系统的行为,摆脱了所有用户进程作为一个整体对系统的要求.显然,对于用户进程相互独立的应用系统,系统功能的相对正确等价于功能正确(所谓进程相互独立,是指进程的状态之间不存在依赖关系).

对于交换系统而言,如果我们将一次呼叫所涉及到的所有用户看成是一个用户进程,并不考虑用户进程之间的资源共享问题,则用户进程之间是相互独立的.因此我们有必要单独对资源管理进程加以分析.用户进程通过资源管理进程相互影响,可以分为两种情况:第 1,影响了分配资源的具体编号,占绝大多数;第 2,影响了资源分配的成功与否,占极小的比率.从用户进程的角度上看,这两种影响均可以忽略.因此,如果我们强行指定这种影响不存在,即,将资源管理进程收到的资源请求消息标记为非重要消息,则可以导出用户进程之间是独立的.

我们的思路是,首先根据应用系统的特点划分用户进程,从概念上初步判断系统功能的相对正确是否已经足够.如果该判断成立,则由定义的系统进程的交互关系推断出导致用户进程依赖关系的进程链和相应的消息.然后逐一对该进程链中的进程进行分析,判断该进程接收相应消息导致的状态变化对另一用户进程的影响,如果认为这种影响可以忽略,则将此消息标记为非重要消息,从而割断用户进程之间的依赖关系.如果没有可以标记的消息,则认为该系统不适合于依赖关系的优化.以上步骤在进程的设计完成之后进行.

系统在恢复时依赖于重要消息形成的依赖关系.

进程 p 保存一个消息编号向量 $M=\langle m_1,m_2,\ldots,m_N\rangle$.其中 m_i 记载进程 i 发给 p 的最近一次的消息编号,m_p 为进程 p 自身的发送消息编号.p 每发送一次消息,m_p 的值加 1,并作为该消息的一部分发送至目的进程.进程在接收到一条消息之后,根据消息内容中附带的消息编号修改自己的消息编号向量值.在正常情况下,从同一进程接收到的连续消息的编号应该是递增的.如果接收到的消息编号小于或者等于向量中对应的消息编号值,则说明消息的发送进程产生了后退.

如果进程 p 在系统的恢复完成之后发现了相关进程的后退,则可以采用前向恢复的思想对进程 p 的状态进

行修正.本文提供一个可能的解决方案.p 暂停正常的处理,启动一个对话.在该对话中,进程向与之具有数据耦合关系的有关进程发送请求消息,收集其状态恢复所必须的数据,直到所有的消息都得到了响应,p 的状态得到修正,对话结束.p 继续正常的处理.对话的启动可以选在一个系统负荷很轻的时间段(对于交换系统而言,可以考虑在凌晨启动对话),从而尽可能减轻状态修正对系统的影响.关于交换系统的有针对性的前向恢复方法,我们将另文给出.

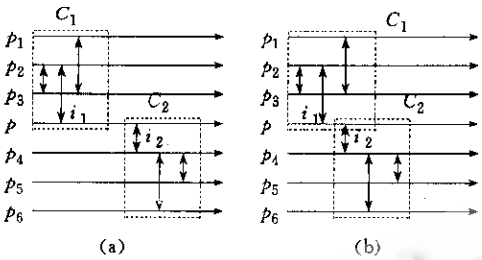

图3

具体的状态恢复算法可以在已有算法(文献[2,5]等)的基础上扩充,这里从略.

我们分析该算法应用于协作的恢复时可能带来的好处.一个典型的会话执行过程如图3(a)所示,进程 p_1,p_2,p_3 和 p 之间的交互构成会话 C_1,进程 p_4,p_5,p_6 和 p 之间的交互构成会话 C_2.从图中可以看出,由于 C_1 和 C_2 存在着共同的参与进程 p,使得 C_1 和 C_2 必须以串行的方式执行.如果对进程可以应用上述算法,则可以按如图3(b)所示的方式执行.其原因在于,由于定义了非重要消息 i_1 和 i_2,使得进程 p 不必遵循随会话内其他进程的回滚而回滚.

在非协作的恢复方法中应用本算法的好处通过图2的说明已经可以看到,这里不再赘述.

3 讨 论

消息语义的概念由 Leong 和 Agrawal 提出,他们首先注意到消息的提交可能不会导致接收进程状态的改变,因此不会引起进程状态对消息的依赖.本文在此基础上做了进一步的扩充,指出针对某些特定的应用进程,即使消息接收进程的状态发生了改变也可以不导致进程状态对消息的依赖性.相比之下,Leong 的方法对非重要消息的判定可以不需要应用进程设计者的参与,而利用静态的数据流分析和动态执行算法的手段来实现;在我们的方法中,非重要消息需要设计人员根据具体应用进程的语义给出.

关于会话的并发性,J. Xu 曾提出了一种称之为"协作的原子动作(CA)"模型[11],该模型将事务处理从会话中分离开来(称为外部原子对象),并认为外部原子对象的并发性控制以及错误恢复应单独加以考虑.本文的思想在某些方面与之有相近之处.

本文的方法是结合应用进程的特点提出的.作者认为,针对特定的应用、特定的数据结构,制定出容忍特定软件错误的方法更切乎实际.

参考文献

1 Randell B. System structure for software fault tolerance. IEEE Transactions on Software Engineering, 1975,1(2):220~232

2 Anderson T, Knight J C. A framework for software fault tolerance in real time systems. IEEE Transactions on Software Engineering, 1983,9(3):355~364

3 Lamport L. Time, clocks, and the ordering of events in a distributed system. Communication of the ACM, 1978,21(7):558~565

4 Birman K, Schiper A, Stephenson P. Lightweight causal and atomic group multicast. ACM Transactions on Computer Systems, 1991,9(3):272~314

5 Johnson D B, Zwaenepoel W. Recovery in distributed systems using optimistic message logging and checkpointing. Journal of Algorithms, 1990,11(3):462~491

6 Wang Y M. Consistent global checkpoints that contain a given set of local checkpoints. IEEE Transactions on Computers, 1997,46(4):456~468

7 Cristian F. Understanding fault tolerant distributed systems. Communication of ACM, 1991,34(2):56~78

8 Leong H V, Agrawal D. Using message semantics to reduce rollback in optimistic message logging recovery schemes. In: Proceedings of the 14th International Conference on Distributed Computing System. Los Alamitors, CA: IEEE Computer Society Press, 1994. 227~234

9 Strom R, Yemini S. Optimistic recovery in distributed systems. ACM Transactions on Computer Systems, 1985,3(3): 204～226

10 Nakamura R, Ono H *et al*. Reliable switching system recovery. In: 1994 IEEE GLOBCOM, NJ: IEEE Inc., 1994. 1596～1600

11 Xu Jie, Randell B, Romanovsky A *et al*. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In: Proceedings of the 25th International Symposium on Fault Tolerant Computing, Digest of Papers. Los Alamitors, CA: IEEE Computer Society Press, 1995. 499～508

A New State Recovery Approach in Distributed System

WANG Zhun[1,2]　CHEN Jun-liang[2]

[1](*Beijing Institute of Southern Telecommunication Development CO. Ltd.　Beijing*　100086)

[2](*National Laboratory of Switching Technology and Telecommunication Networks*
Beijing University of Posts and Telecommunications　Beijing　100876)

Abstract　In the distributed system, all processes that depend on the process, which must be rolled back, have to be rolled back as well in order to keep the global state consistency. The dependency relation defined in the previous literature is too restrict. A new definition of dependency relation is proposed in this paper, and the influence of a process' rolling back can be reduced.

Key words　Software fault tolerant, consistent state recovery, recovery blc