一种基于信道不可靠环境的协调式检查点协议

张杰智,任国林
(东南大学 计算机科学与工程学院,江苏 南京 210096)

摘 要:在分布式计算环境中经常使用检查点/恢复策略来进行容错。文中主要研究在信道不可靠的环境中通过协调使相互通信的各进程所做的检查点保持全局一致性的方法。通过分析中途消息与信道可靠性之间的关系以及已有检查点协议对于中途消息处理方法,提出了一种应用于信道不可靠环境下的协调式检查点方法,其消息复杂度为 $O(N)$ 且不引入其他的计算负担,只通过一次同步即可达到全局一致性状态,相比于以往的协调式检查点协议大大减小了时间开销,提高了在不可靠信道环境中做全局一致检查点的效率。

关键词:检查点协议;全局一致性;协调式检查点
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)02-0055-04

A Coordinated Checkpointing Protocol Based on Unreliable Channels

ZHANG Jie-zhi, REN Guo-lin
(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: In distributed computing environments, checkpoint/rollback recovery mechanism is widely used to achieve fault tolerance. Focuses on a globally consistent checkpoint protocol developed for distributed processes that communicate with each other through unreliable channels. After analysing the methods adopted by existing global checkpointing protocols dealing with in-transmit messages, bring forward a novel method that is applicable in unreliable communication channel. The message complexity involved is within $O(N)$, with no other related computation overhead. It can achieve global consistency through one synchronization, thus greatly reduces the time overhead comparing with the existing methods. Tests indicate that it performs over existing mechanism of channel clearing or message counting.

Key words: checkpointing protocol; global consistency; coordinated checkpoint

0 引 言

在大规模集群计算环境中,许多大型程序运行时间较长,有的需要几天、几周甚至几个月。由于机群自身的特点,随着其计算结点数量的增加,结点失效率也不断增高。减小节点失效带来的计算损失成为机群技术研究中很重要的一项课题。目前最常用的是用设置检查点的方法来容错。所谓检查点,是指在程序正常运行过程中适时地记录进程的各种信息并保存到可靠的存储介质中,在结点出现故障后可以根据所保存的信息重新恢复进程当时的状态,从而能够有效地减少计算损失。

在多进程并行程序中,进程间常常会有消息传递,这种情况下只是独立地保存进程本身的信息是不够的。为了保证应用程序能够正常恢复,各进程的检查点设置必须符合全局一致性,即要做全局检查点。

全局检查点按其检查点协议主要分三种[1]:独立式检查点、消息驱动式检查点[2]和协调式检查点。

(1)独立式检查点是各进程独立进行检查点设置,各进程之间不存在协调。这种检查点协议的优点是做检查点非常方便,没有协调开销。缺点是在恢复时可能产生多米诺效应,应用程序可能需要卷回到初始状态,从而浪费了大量的计算,实用价值不高。

(2)消息驱动式检查点,要求进程在正常计算中发送消息时,附加一些跟检查点有关的协调信息,各进程根据这些附加的信息有选择地做检查点,从而避免多米诺效应,保证了全局一致性。这种方式的缺点有二:第一是可能会有多余的检查点,这种检查点不属于任何全局检查点,做这部分检查点的开销是完全没有必要的;第二是检查点协议和恢复协议都比较复杂,开销比较大。

(3)协调式检查点是在做检查点前,通过对所有进程进行协调来保证全局一致性。其主要优点是检查点协议和恢复协议都比较简单,并且不会做多余的检查

收稿日期:2007-07-14
作者简介:张杰智(1983-),男,山西交城人,硕士研究生,研究方向为计算机体系结构、高性能计算;任国林,副教授,研究方向为计算机体系结构、嵌入式系统、高性能计算。

点,其恢复开销是各种协议中最小的。主要缺点是每次同步都要求所有的进程参加,导致较大的协调开销。

在信道不可靠环境下,应用程序需要自行解决消息丢失的问题,程序中常用的处理方法是消息接收方在收到消息后,发送一个确认消息到发送方,发送方收到确认消息后认为该次通信成功,这个确认消息通过应用程序、非底层通信协议实现。对于那种无关紧要的消息,应用程序甚至可以直接忽略之。在这种特殊的应用环境下,为这类应用程序做检查点的时候就可以不考虑中途消息,从而可以减少保证全局一致性所需要进行的同步次数。对于协调式检查点协议来说,同步次数减少意味着协调开销的显著降低。基于此应用特征提出了一种基于不可靠信道环境的协调式检查点协议——一次同步协议,与传统的协调式检查点方法相比能够大大减小同步开销,从而提高不可靠信道环境下做全局一致检查点的效率。

1 系统模型

1.1 主要概念

假设分布式程序 P 由 N 个进程 $P_1, P_2, \cdots, P_N$ 组成,$C_{i,m}$ 为进程 P_i 的第 m 个检查点文件。一个进程间消息 M 有两个与之相对应的事件:消息发送事件 send(M) 和消息接收事件 receive(M)。当进程 P_i 发送消息 M 的时刻早于进程 P_i 做第 m 个检查点的时刻,称 send(M) $\in C_{i,m}$;当进程 P_i 接收消息 M 的时刻早于进程 P_i 作第 m 个检查点的时刻,称 receive(M) $\in C_{i,m}$。

定义 1 设集合 $S=(C_{1,m_1}, C_{2,m_2}, \cdots, C_{N,m_N})$ 是分布式程序 P 各进程的检查点文件组成的程序全局状态。若 $\forall$ 消息 M 和 $\forall$ 整数 $i(1 \leqslant i \leqslant N)$ 都有:receive(M) $\in C_{i,m_j} \Rightarrow \exists j(1 \leqslant j \leqslant N)$send($M$) $\in C_{j,m}$,则称集合 S 为一个一致的全局状态[3]。文中,集合 S 称为一个全局检查点。

定义 2 $\forall$ 消息 M,若 receive(M) $\notin C_{i,m}(1 \leqslant i \leqslant N)$ 且 send(M) $\notin C_{j,m}(1 \leqslant j \leqslant N)$,则称 M 为孤儿消息[4]。

定义 3 $\forall$ 消息 M,若 receive(M) $\in C_{i,m}(1 \leqslant i \leqslant N)$ 且 send(M) $\in C_{j,m}(1 \leqslant j \leqslant N)$,则称 M 为中途消息[4]。

如图 1 所示,在全局状态 C_2 中,m_1 和 m_2 都是中途消息;在全局状态 C_3 中,m_3 是孤儿消息。

过去所提出的协调式检查点机制是基于保存消息计数[5]或消息驱赶[1]等方式,消息发送方在发送某消息结束后做一个检查点,只有当信道中的消息到达接收方后,消息接收方进程才能开始做检查点。否则会出现定义 3 所描述的中途消息,如图 1 所示的 m_1, m_2 消息,因而导致了较高的协调开销。

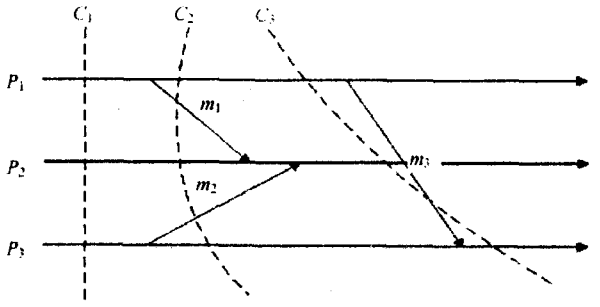

图 1 全局一致性检查点与非全局一致性检查点

在信道不可靠环境中,中途消息与由于信道不可靠而导致的消息丢失在应用程序看来没有区别。中途消息的存在,不会影响检查点集合的全局一致性[1]。文中提出的协议是基于信道不可靠环境的假设,应用程序必须自行解决消息丢失的问题(程序重发或直接忽略丢失的消息),所以中途消息可以放在文中提出的检查点/恢复机制的考虑之外。文中所提出机制的主要特点就在于舍弃这些中途消息,而只针对消除孤儿消息作一次同步,减少了不必要的同步开销,使同步消息数在 $O(N)$ 量级,并且与已有的 $O(N)$ 消息复杂度的检查点机制相比,减少了由维护消息计数导致的额外计算和通信开销。

1.2 系统模型假设

文中提出的一次同步协调式检查点协议主要用于对由 N 个相互通信的进程组成的分布式应用程序进行检查点操作,协议针对具有如下特征的系统:

a) 各进程间无共享存储器,所有进程间通信均通过消息传递机制进行;

b) 系统没有全局统一时钟;

c) 通信信道是不可靠的,可能丢失消息,应用程序应能够自行解决消息丢失问题;

d) 不要求通信信道是 FIFO 的。

2 一次同步协议

2.1 协议算法描述

本算法中使用了一个协调者进程 Manager,通过它与各计算进程之间传递同步消息来使各计算进程达到全局一致性。下面有时会将计算进程简称进程,Manager 进程简称 Manager。协议总体同步过程如图 2 所示。具体步骤如下:

开始做检查点时,Manager 首先给每个计算进程发 init 消息通知做检查点。各计算进程一旦收到这个 init 消息马上停止发送后续消息,立即开始做检查点,

并发送反馈消息 ack 给 Manager 表示已经接到通知。为保证协议的严密性,开始做检查点时间要早于发送 ack 消息时间。Manager 发完 init 消息后等待各计算进程的 ack 消息。当所有计算进程都至少有一个 ack 消息被收到后,Manager 再发通知消息 resume 给所有计算进程,表示已经协调完所有进程,各进程可恢复正常运行。

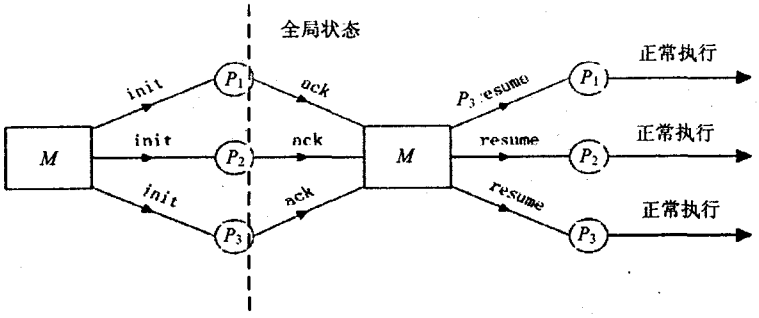

图 2　Manager 进程与计算进程之间的通信

各计算进程收到 init 消息时,无论之前最近的消息动作是接收还是发送,都不会破坏全局一致性。如果消息动作是发送,那么在发送以后做检查点本来就不会破坏一致性,所可能导致的中途消息由计算进程的消息丢失处理策略得到解决;如果消息动作是接收,说明消息的发送方尚未做检查点,即发送方尚未收到 init 消息,该消息不是孤儿消息。

算法流程及时序如图 3 所示。

Manager
空闲;
MS1 向各计算进程广播 init 消息,
　等待接收各进程的 ack 消息
MS2 接收到所有进程发来的第一个 ack 消息后,
　向各进程广播 resume 消息;
　若到达最大等待时间,还有未收到其 ack 消息的进程
　　则返回"检查点失败"并终止该次检查点
MS3 空闲
Pi
正常计算;
PS1 若收到 init 消息,
　　置 Send_Count = 0,
　　挂起计算进程,开始做检查点(fork)
　否则转 PS4;
PS2 向 Manager 发送 ack 消息,Send_Count + +;
　等待接收 resume 消息(之前不会向其他进程发送消息)
PS3 若收到 resume 消息,转 PS4;
　若超过最大等待时间,还未收到 resume 消息,
　　若 Send_Count≤最大重发次数,转 PS2,
　　否则撤销所做检查点文件
PS4 正常计算

图 3　一次同步协议算法流程及时序

Manager 进程本身并不做检查点,即其状态不属于全局状态,在与 Manager 同步过程中,进程发给 Manager 的消息就可能会成为中途消息,因为消息的发送事件可能会包含在某个全局状态中(如做下一轮检查点时的全局状态),而其接收事件却一定不包含在这个全局状态中。在文中不可靠信道环境的假设下,解决这个问题的办法是极其简单的。在进行卷回恢复时这种中途消息的产生可以安全地归结为由信道不可靠造成的消息丢失。恢复协议只需要忽略这些消息即可,因为它不会影响计算结果,也就是说由于与 Manager 进程通信而产生的这种中途消息不会对检查点恢复效果有任何的影响。

若某消息已被接收进程所收到,则该消息的发送动作必已经完成。而已发送但尚未接收到的中途消息可归结为由不可靠信道造成的消息丢失去进行处理。因此文中算法不需要信道保证 FIFO。

2.2　算法正确性证明

结论 1　一次同步算法可在有限时间内终止。

证明:在做检查点的同步过程中,算法为 Manager 进程和计算进程(P_i)均设置了最大等待时间,同步超时会导致检查点失败,进程继续执行。故文中算法可在有限时间内终止,而不受结点故障、网络故障等影响。

结论 2　一次同步算法的消息复杂度是 $O(N)$。

证明:Manager 所发 init 消息和 resume 消息面向所有进程,共需发 $2N$ 个消息。各计算进程均需发一次 ack 消息,总数为 N 个。文中基于不可靠信道环境,故通过重发机制保证检查点同步过程的有效性,因重发次数有限且为常数,所以检查点同步过程的消息复杂度是 $O(N)$ 量级的。

结论 3　一次同步算法所得到的检查点集合满足全局一致性条件。

证明:利用反证法证明。

假设通过一次同步算法得到的检查点集合不满足全局一致性条件,则必存在一个孤儿消息 M。若 M 的发送者为进程 P_k,接收进程为 P_l,则有 receive(M) $\in P_l$ 且 send(M) $\notin P_k$。由 send(M) $\notin P_k$ 可知消息 M 的发送必在 P_k 收到 resume 消息之后,又因 receive(M) $\in P_l$ 可知消息 M 的接收必在 P_l 收到 init 消息之前。根据算法要求,P_k 收到 resume 消息之时,P_l 肯定已收到 init 消息。由此得出消息 M 的接收动作发生在其发送动作之前,二者是不可能的。

因此假设不成立。得证。

3 协议验证

3.1 验证思想

验证的主要目的是比较文中算法与已有协议的通信开销。由于现有检查点软件中没有设计协调式检查点协议的完整接口,而文中考虑的重点仅限于通信协议,故文中实验基于 MPI 并行环境[6]编程模拟各种协议来进行通信开销比较,而不是将协议实现在检查点软件中再进行比较。

比较的侧重点在消息驱赶机制(MsgClear)、消息计数机制(MsgCount)、文中所提一次同步机制(OnceSync)三种协议的协调开销方面。

3.2 实验及结果

实验使用基于 MPI 的 1024 * 1024 数值矩阵的 Jacobi 叠代程序,迭代 4000 次,每次叠代完成后都调用一次协调协议,迭代过程中的矩阵分割数为参与协调协议的进程数(包括 Manager)。因通信是对称的,故可使用各进程实际执行时间的平均值来代表协议的开销。实验结果如表 1 及图 4 所示。

表 1 三种协议通信开销实验结果

进程数	一次同步	消息驱赶	消息计数
2	113.58	114.80	117.47
4	53.91	54.28	55.07
8	96.13	119.20	131.37
16	193.19	273.68	321.40

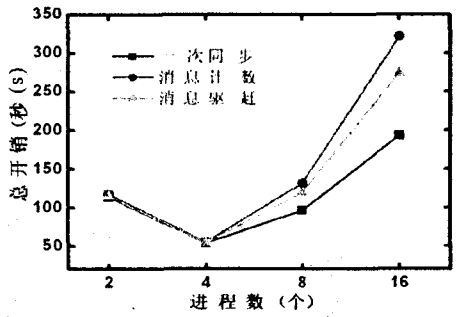

图 4 三种协议通信开销比较曲线图

由图 4 可看出,消息计数协议由于其附加信息的计算和记录、发送开销而导致性能较差;消息驱赶协议由于每条通信信道上都需要发一个驱赶消息,消息复杂度为 $O(N_2)$[5],性能也不理想;文中所提一次同步协议的消息复杂度为 $O(N)$,且没有引入其他的计算和记录开销,所以在各种叠代次数下性能都优于其他两种机制。

4 总 结

提出了一种应用于通信信道不可靠环境下的开销较小的协调式检查点协议——一次同步协议,其主要依据是在不可靠信道环境中,中途消息和由于信道不可靠而导致的消息丢失在用户应用程序看来是没有区别的。通过把中途消息当作消息丢失交给用户程序去处理,减少协调所需要的开销。实验证明它比传统的协议有更好的性能。但因其要求应用程序本身具有消息丢失的处理机制,对该协议的应用范围构成了一定的限制。

参考文献:

[1] Elnozahy E N, Alvisi L, Wang Y M, et al. A Survey of Rollback-Recovery Protocols in Message-Passing Systems[J]. ACM Computing Surveys, 2002, 34(3): 375-408.

[2] Alvisi L, Rao S, Husain S A, et al. An Analysis of Communication-Induced Checkpointing[C]// Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing. Washington, DC., USA: IEEE Computer Society, 1999.

[3] 魏晓辉,鞠九滨. 分布式系统中的检查点算法[J]. 计算机学报, 1998(4): 367-375.

[4] Helary J M. Communication-Induced Determination of Consistent Snapshots[J]. IEEE Transactions on Parallel and Distributed Systems, 1999, 10(9): 856-877.

[5] 汪东升,邵明珑. 具有 O(n)消息复杂度的协调检查点设置算法[J]. 软件学报, 2003, 14(1): 43-48.

[6] The MPI Forum. The MPI Message-passing Interface Standard[EB/OL]. 1995-05. http://www.mcs.anl.gov/mpi/standard.htm.

(上接第 54 页)

[5] Hjaltason G R, Samet H, Sussmann Y J. Speeding up Bulk-Loading of Quadtrees[C]// In ACM International Workshop on Advances in Geographic Information Systems. Indiana, USA: Addison-Wesley, 1997: 50-53.

[6] Bayer R, Markl V. The UB-Tree: Performance of Multidimensional Range Queries[R]. Germany: Institut f"ur Informatik, TU Munchen, 1997.

[7] Paller A. Rely on Red Brick's Performance for Data Warehouse Applications[R]. Massachusetts: DataWarehousing Institute, Informix Corporation, 2000.

[8] Orenstein J A, Merrett T H. A Class of Data Structures for Associative Searching[C]// In Proceedings of the Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. Waterloo, Ontario, Canada: ACM, 1984: 181-190.