一种可靠高效的回卷恢复实现方法

杨金民, 张大方, 黎文伟
(湖南大学软件学院, 湖南长沙 410082)

摘 要: 本文针对现有用户级进程检查点实现中的线程挂起点不确定性问题提出一种基于线程自挂的解决方案. 另外, 为了降低分布式回卷恢复开销, 本文提出一个多线程化的回卷恢复实现基架. 基于所提回卷恢复策略, 开发了一个回卷恢复试验床 WINDAR. 试验结果表明, 多线程化实现策略能够显著提高悲观消息日志协议性能.

关键词: 检查点; 进程状态; 回卷恢复; 多线程
中图分类号: TP302.8 文献标识码: A 文章编号: 0372-2112 (2006) 02-0237-04

A Robust and Efficient Rollback Recovery Implementation Scheme

YANG Jin-min, ZHANG Da-fang, LI Wen-wei
(Software School of Hunan University, Changsha, Hunan 410082, China)

Abstract This paper addresses the issue of nondeterminacy of thread suspension point in user-level process checkpointing, and proposes a solution to it. In addition, the workloads of a process are multithreaded in terms of rollback recovery to exploit effectively system resources for reducing the overhead. The presented schemes are implemented in rollback recovery testbed WINDAR. Experimental results show that the presented scheme improves the performance of rollback recovery, especially for pessimistic message logging protocols.

Key words checkpointing; process state; rollback recovery; multithreading

1 引言

回卷恢复技术[1]是提高系统可靠性的一种常用方法, 它在容错计算、并行软件和长时间运行软件的调试、负载平衡、移动计算、系统安全等领域得到了广泛应用. 回卷恢复中, 回卷恢复系统的健壮性和回卷恢复开销一直是人们关注的重点. 回卷恢复本身复杂, 回卷恢复组件在提高系统可靠性的同时, 也使系统更加复杂, 隐藏更多隐患. 研究回卷恢复基架和可重用组件, 以简化应用程序开发中回卷恢复功能的实现, 目前已有许多成果. 就进程检查点的用户级实现, 已有许多检查点运行库[2~7]; 分布式回卷恢复实现有 Egida[8]. 这些研究成果为回卷恢复技术的应用提供了基础. 不过现有的进程检查点用户级实现, 在某些情况下可能出现恢复不成功问题, 不够健壮. 现有分布式回卷恢复实现, 因其开销大往往不能满足应用要求. 本文针对现有进程检查点用户级实现的不足, 提出一种基于线程自挂的检查点方法, 以提高进程检查点系统的可靠性. 对分布式回卷恢复实现, 本文提出一种基于多线程化的实现基架, 利用多线程技术来减小回卷恢复开销.

2 基于线程自挂的检查点实现方法

2.1 现有用户级实现方法的问题

进程状态包括用户空间状态和进程环境两部分. 进程用户空间状态和进程环境存在耦合关系, 即用户空间状态中存有进程环境中对象的引用. 在用户级做进程检查点, 进程环境不能直接读取, 也不能通过拷贝方式恢复. 对该问题, 现有解决方法是: 跟踪进程对 API的调用, 然后基于跟踪信息来判定进程环境. 当发生故障时, 系统重启, 恢复系统使用检查点信息来恢复进程状态, 使进程从检查点状态继续执行. 进程环境的恢复是使用检查点信息通过 API调用来完成. 进程环境不能严格一致恢复, 恢复后用户空间状态和进程环境的耦合关系有可能被打破. 针对该问题, 句柄重现方法[4]在恢复时反复调用句柄复制 API复制对象句柄, 直至系统返回一个相同的对象句柄, 以保持正确的耦合关系. 该方法的问题是: 某些情况下有可能复制不出原有句柄. 虚拟句柄策略[6,7]是另外一种解决方法: 应

收稿日期: 2004-06-25; 修回日期: 2005-10-18
基金项目: 国家自然科学基金 (No. NSFC 60473031); 湖南省自然科学基金 (No. 05JJ30116)

用程序使用虚拟句柄访问内核对象, 检查点系统负责虚拟句柄与实际句柄的转换. 由于做检查点时挂起应用线程是一种异步操作, 不能控制挂起点的位置, 该方法依然存在恢复不成功的隐患. 图 1为恢复不成功的例子. 图 1中假定应用程序被挂起在 SetFilePointer这个 API的代码中. 故障后, 重启恢复执行会有如下两个问题: (1)使用原来的文件句柄 (h = 1)去访问恢复后的内核文件对象 (由于不能严格一致恢复, 句柄 h = 3), 执行失败; (2)即使恢复后的没有句柄问题, 恢复时使用检查点信息把文件指针恢复为 p = 5, SetFilePointer返回后, p = 8, 而实际应该等于 5, 产生逻辑错误.

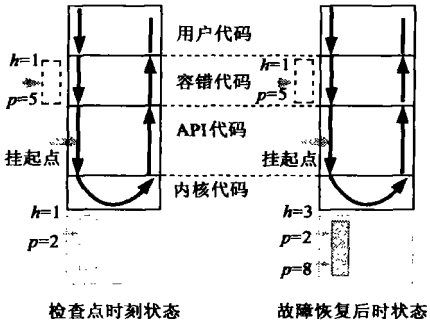


图 1 现有检查点策略存在的问题

2.2 基于线程自挂的检查点策略

从上述问题可知, 要保证恢复执行成功, 做检查点时, 容错系统中记录的内核对象的属性值必须与内核对象的实际值保持一致; 应用线程的栈中不能有内核对象的实际句柄. 要实现这两点, 在做检查点时, 容错代码、API代码和内核代码必须以原子方式执行, 即应用线程的挂起点不能位于这些代码中.

我们采用应用线程自己挂起自己的策略来实现容错代码、API代码和内核代码以原子方式执行. 具体过程如下:

在检查点时刻到点时, 检查点线程设置全局检查点标志 Checkpoint为 ON, 此时应用线程可分为运行态线程和等待态线程.

对运行态线程, 它可分两种情况: (1)正在执行用户代码段 (图 2所示) (判别方法为其 InAPI为 OUT); (2)正在执行容错代码或 API代码或内核代码 (判别方法为其 InAPI为 IN).

(1)对 InAPI为 OUT的应用线程, 检查点线程对其执行挂起操作. 其挂起点位置又有两种情况: (a)挂起点位于用户代码中; (b)挂起点位于检查点例程 CCS1中 (图 2所示).

(a)当挂起点位于用户代码中时, 检查点线程为其建立实时中断例程 CIS(图 2所示), 修改其上下文, 然后恢复其执行, 使其执行转入中断例程. 在中断例程中应用线程调用检查点例程, 释放其拥有的同步对象, 然后挂起自己.

(b)当挂起点位于 CCS1时, 检查点线程恢复其执行. 在 CCS1中应用线程执行检查点例程, 释放其拥有的同步对象, 然后挂起自己.

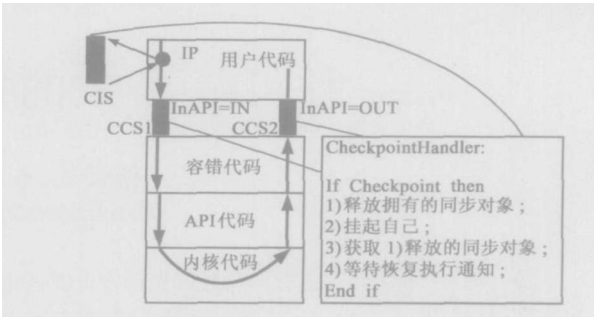


图 2 基于线程自挂的检查点方法状态

(2)对正在执行系统代码段的线程, 当其执行到 CCS2中检查点例程时, 释放其拥有的同步对象, 然后挂起自己.

运行态应用线程释放其拥有的同步对象, 并挂起自己后, 在等待态的应用线程会结束等待, 当其执行到 CCS2中检查点例程时, 它释放其拥有的同步对象, 然后挂起自己.

在所有应用线程都挂起自己后, 检查点线程做进程状态快照. 快照做完后, 检查点线程恢复所有应用线程的执行, 应用线程恢复拥有其在检查点例程中释放的同步对象, 之后等待检查点线程的通知. 在所有应用线程都恢复拥有其原有的同步对象之后, 检查点线程通知所有应用线程恢复执行. 对运行态线程, 它们恢复正常执行; 对等待态线程, 重新调用等待 API函数进入等待状态.

3 多线程化的回卷恢复基架

尽管分布式回卷恢复协议多种多样, 但它们都必须实现输出提交的状态间隔不会被回卷, 以及故障恢复之后, 系统无孤儿消息. 每种协议都不外乎一些基本组件. 这些基本组件是: 依赖跟踪组件, 事件日志组件, 输出提交组件, 检查点组件, 故障检测组件, 故障恢复组件, 消息重放组件, 垃圾回收组件. 基于回卷恢复协议的共性和这些基本组件可以建立基于事件驱动的回卷恢复统一基架 (图 3所示).

多线程是一种提高计算资源利用率和提供轻量级并发处理的有效方法. 当处理器带有超线程技术时, 多线程化应用程序的工作负载显得尤为重要. 多线程化程度越高, 从处理器挖掘出的计算潜力也就越大.

进程的执行可模型化为事件驱动的执行过程. 进程执行中有 5类事件: 检查点事件, 日志事件, 故障恢复事件, 回卷事件, 输出提交事件. 在设计回卷恢复系统时, 我们通过建立消息缓冲队列来多线程化进程的工作负载. 消息缓冲队列建立之后, 计算, 接收消息, 发送消息, 消息日志, 保存检查点可以并发执行, 从而减少回卷恢复开销. 除了计算线程之外, 回卷恢复系统中另外创建了五个线程, 它们分别是: 回卷恢复线程, 消息发送线程, 消息接收线程, 存储访问线程, 和联接请求监听线程. 线程间的控制流和数据流如图 4所示.

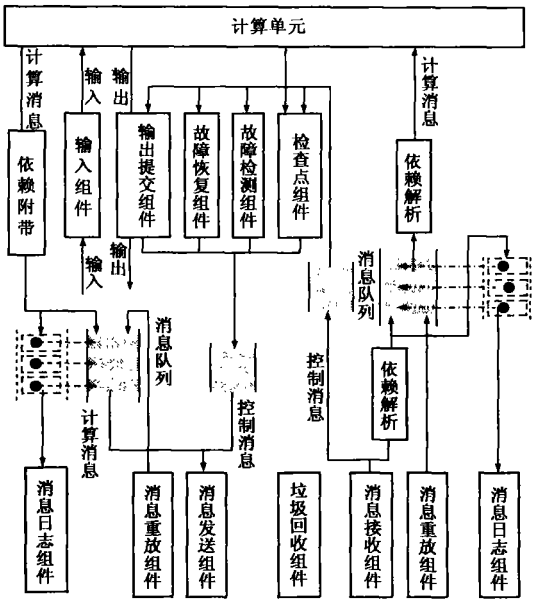


图 3　分布式系统中进程回卷恢复框架

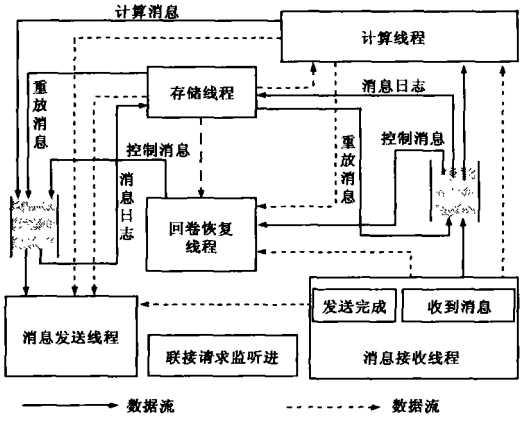


图 4　多线程并发执行模式

4　试验结果

基于上述检查点方法和多线程化的回卷恢复基架, 我们在 W indow s平台上实现了一个回卷恢复试验床 W IN-DAR[2]. 我们采用 SPLASH-2标准性能测试码 Barnes和 W ater-nsquared 在系统环境动态变化的 PC 机上来测试恢复执行正确情况, 以及检查点库运行时开销和进程检查点开销. 图 5和表 1以为试验结果. 从图 5和表 1可看出, 基于线程自挂的检查点实现方法解决了系统环境变化情况下恢复执行不成功的问题, 而检查点系统运行时开销增加微乎其微.

表 1　运行库开销和检查点开销对比

测试程序	W illiam 的结果[3]		W indar结果	
	API包裹开销%	一个检查点开销	API包裹开销%	一个检查点开销
barnes	2.0	≈ 0.2Sec	2.4	≈ 0.21Sec
W ater-nsquared	0.6	≈ 0.3Sec	0.7	≈ 0.27Sec

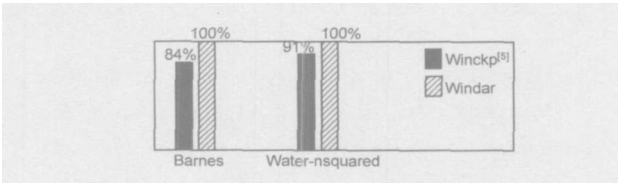


图 5　恢复成功率

为了测试多线程化回卷恢复实现策略对提高回卷恢复性能的有效性, 我们把 Egida[8] 移殖改造到 w indow s平台上. Egida是一个单线程回卷恢复系统. 我们采用三个 NPB2.3标准性能测试程序 LU, BT和 SP来测试回卷恢复性能参数, 试验结果如图 6所示. 图 6结果表明多线程化进程工作负载能够取得更高的执行效率. 多线程模式与单线程模式相比, LU, BT 和 SP 的执行时间分别减少 1.4%, 9.0%, 19.2%. 值得注意的是多线程并发机制显著地减少了悲观消息日志的开销. LU, BT 和 SP在单线程模式下的悲观消息日志开销分别为 18.9%, 3.9%, 8.7%, 而在多线程模式下其悲观消息日志开销分别为 6.1%, 0.6%, 2.4%. 详细原因分析见 W indar[2].

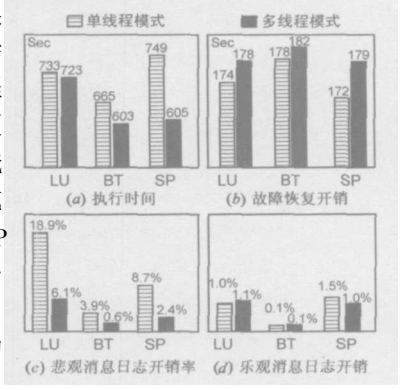


图 6　单线程模式和多线程模式下回卷恢复性能对比

5　结论

现有用户级进程检查点实现中, 应用线程的挂起点具有不确定性, 当挂起点位于系统 API代码或者内核代码中时, 可能引起恢复失败或恢复执行不正确. 本文提出基于线程自挂的实现方法, 在原有的异步控制方法中加入延缓机制, 实现容错代码、系统 API代码和内核代码在做检查点时以原子方式执行, 保证了恢复执行正确. 另外, 本文提出一个多线程化和组件化的分布式回卷恢复实现基架, 解析进程负载, 并使之并发执行, 降低回卷恢复开销. 基于所提回卷恢复实现方法, 我们开发了一个回卷恢复系统 W indar. 试验结果表明, 我们的回卷恢复实现策略不仅能保证恢复执行正确, 而且能够有效地降低回卷恢复开销, 对悲观消息日志协议, 效果尤其明显.

参考文献:

[1] Elnozahy E, et al. A survey of rollback recovery protocols in message passing systems[J]. ACM Computing Surveys, 2002, 33(3): 375-408.

[2] Yang JM, et al. W INDAR: A multithreaded rollback-recovery toolkit on w indow s[A]. 10th IEEE Pacific Rim

dependable Com-puting Symp. [C]. Tahiti, 2004. 395-400.

[3] Dieter W R, et al. A user-level checkpoint-ing libraray for POSIX threads programs[A]. 29th Symp on Fault-tolerant Computing Systems[C]. Madison, 1999. 224-227.

[4] Sroujii J, et al. A transparent checkpoint facility on NT [A]. 2nd USENIX Windows NT Symp[C]. Seattle, 1998.

[5] Chung P E, et al. Winckpt: A transparent checkpointing and rollback recovery tool for windows NT applications [A]. 29th Symp. on Fault-tolerant Computing[C]. Wisconsin, 1999. 220-223.

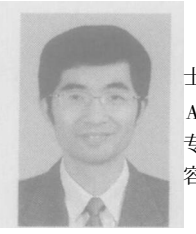
[6] 张悠慧,汪东升,郑伟民. WinNT环境下的进程检查点设置与回卷恢复[J]. 计算机研究与发展, 2001, 38(1): 50-55.

[7] R Nasika, et al. Transparent migration of distributed communicating processes[A]. Proc. of PDCS 2000 [C]. Las Vegas, 2000.

[8] S Rao, et al. Egida: An extensible toolkit for low-overhead fault tolerance[A]. 29th Symp. on Fault-Tolerant Computing[C]. Madison, 1999. 48-55.

作者简介:

杨金民 男, 1967年 12月生于湖南宁乡县. 博士, 副教授. 主要研究兴趣有软件容错, 软件工程, 系统可靠性.
E-mail: yjmswf@vip.136.com.



张大方 男, 1959年 4月出生于上海. 博士, 教授, 博士生导师. IEEE WRTLT和 IEEE ATS指导委员会委员, 中国计算机学会容错计算专委会副主任, 全国政协委员. 主要研究方向有容错计算, 网络测试, 软件工程, 系统可靠性.

黎文伟 男, 1975年 9月生于湖南沅江. 湖南大学博士生. 研究方向为网络测试, 系统可靠性.