软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(9):2248−2262 [doi: 10.3724/SP.J.1001.2011.03937]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于事务回退的事务存储系统的故障恢复*

宋 伟+, 杨学军

(国防科学技术大学 计算机学院 并行与分布处理国家重点实验室,湖南 长沙 410073)

Fault Recovery Based on Transaction Rollback in Transactional Memory

SONG Wei+, YANG Xue-Jun

(National Key Laboratory of Parallel and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: frank1997@gmail.com

Song W, Yang XJ. Fault recovery based on transaction rollback in transactional memory. *Journal of Software*, 2011,22(9):2248−2262. http://www.jos.org.cn/1000-9825/3937.htm

Abstract: This paper addresses the issue of fault tolerance in transactional memory, and proposes a new method of fault recovery based on transaction rollback (FRTR). The method achieves an efficient fault recovery in transactional memory by utilizing the data-versioning mechanism of transactional memory to avoid the extra overhead of saving the checkpoint data. This paper provides the correctness of this method by proving the isolation of the fault tolerant transactional memory. Finally, this paper presents the design of the FRTRs for 5 test programs, including 4 SPLASH-2 benchmarks. The experimental results show compared with the checkpointing mechanism, the FRTR avoids the extra overhead of saving the checkpoint data and has a low overhead of the fault recovery.
Key words: fault tolerance; fault tolerant transactional memory; fault recovery; transaction rollback; isolation

摘 要: 针对事务存储系统机制下的容错问题,提出一种基于事务回退的事务存储系统的故障恢复方法.该方法利用事务存储系统自身的版本管理机制,避免了额外的检查点数据保存开销,从而实现了事务存储系统高效的故障恢复.通过对容错事务存储系统的隔离性证明了该方法的正确性.最后,使用包括 4 个 SPLASH-2 典型用例在内的 5 个测试程序对该方法进行了性能测试.实验结果表明,与经典的 Checkpointing 机制相比,该方法在避免了额外的检查点数据保存开销的同时,还具有较低的故障恢复开销.
关键词: 容错;容错事务存储系统;故障恢复;事务回退;隔离性
中图法分类号: TP316 **文献标识码**: A

随着微处理器的发展由传统的单核高主频转向共享存储的多核化结构,开发线程级并行受到了越来越多的关注.传统的多线程编程模型(如基于锁机制的多线程编程模型)在避免死锁和开发高可扩展性程序时显露出的复杂、易错等弊端越来越明显.而与此相对,事务存储(transactional memory,简称 TM)以其具有的易编程性和高可扩展性,作为一种有前途的用于解决多核处理器并发访问共享数据的机制,越来越多地受到了研究者的关

* 基金项目: 国家自然科学基金(60921062, 60633050)
收稿时间: 2010-05-10; 修改时间: 2010-07-28; 定稿时间: 2010-08-27
CNKI 网络优先出版: 2011-03-16 11:30, http://www.cnki.net/kcms/detail/11.2560.TP.20110316.1130.002.html

注[1,2].另一方面,近年来多核处理器大量进入高性能计算机系统,仅 2009 年 11 月发布的 Top 500 高性能计算机系统中,前 5 位系统中除第 2 位以外都采用有同构多核处理器[3],这使得事务存储进入高性能计算机系统成为可能.随之而来的,高性能计算机系统固有的可靠性问题使得事务存储机制下的容错问题也将逐渐成为一个值得关注的问题.

目前,针对事务存储系统的容错研究几乎没有起步,而经典的 rollback-recovery 容错技术应用于事务存储系统缺乏针对性,容错开销较大.本文提出了一种基于事务回退的事务存储系统的故障恢复方法(fault recovery based on transaction rollback,简称 FRTR).这种方法充分发挥事务存储系统自身的容错特性,利用事务存储系统本身的、用以保证事务执行的原子性的数据版本管理机制进行事务存储系统的故障恢复,因此避免了经典的 rollback-recovery 容错技术由于额外的检查点数据保存而带来的高开销,并实现了高效的事务存储系统的故障恢复.

本文首先介绍了 FRTR 的基本思想,提出了基于 FRTR 的容错事务存储系统,进而通过对容错事务存储系统的隔离性进行系统的讨论和证明,证明了 FRTR 方法的正确性.本文最后通过包括 4 个 SPLASH-2(stanford parallel applications for shared-memory)测试程序[4]在内的 5 个测试程序对本文的方法进行了性能测试,并与经典的 checkpointing 机制进行了比较.实验结果表明,与经典的 checkpointing 机制相比,本文的 FRTR 方法在无故障运行时几乎不增加额外的运行开销,在有故障时也具有较低的故障恢复开销.

本文第 1 节详细描述基于事务回退的事务存储系统的故障恢复方法.第 2 节讨论并证明基于 FRTR 的容错事务存储系统的隔离性,从而证明 FRTR 方法的正确性.第 3 节通过实验对本文的方法进行性能评估,并与经典的 checkpointing 方法进行性能对比.第 4 节回顾相关工作.第 5 节对本文的工作进行总结.

1 基于 FRTR 的容错事务存储系统

目前的计算机系统中,存储系统中一般具有诸如 ECC 之类的校验保护机制.因此,本文假设存储系统是可靠的,并假设故障只发生在计算部件中.此外,由于本文仅针对事务存储系统的故障恢复机制进行研究,不对故障检测机制进行设计,故假设系统在每个事务提交之前存在有合适的故障检测机制可以检测出系统运行时所产生的瞬时故障.同时,假设程序的运行环境是一个纯事务存储系统,也就是系统中原子事务是基本的并行工作、通信单位,不包含非事务操作.

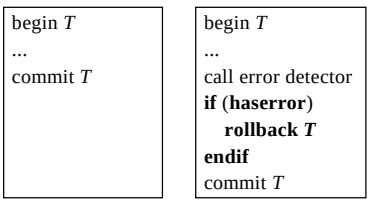
1.1 FRTR基本思想

经典的 rollback-recovery 技术依赖于检查点的保存[5],额外的检查点保存引入系统无故障时的运行开销,而在故障时常常需要多个进程同时回退以保证系统状态的一致性,致使无故障结点上的有效计算工作被浪费.

事务存储系统为保证事务执行的原子性,提供了特定的数据版本管理机制.通常有两种方法来实现这种数据版本管理机制:一种是基于日志的,计算结果直接写入存储单元,同时记录必要的日志信息用以取消事务操作;另一种是基于缓存的,计算结果缓存在高速缓存中直至事务提交,当事务取消时将缓存清空.

事务存储系统的这种用以保证原子性的数据版本管理机制蕴含了一种类检查点的概念,可以说,事务存储系统具有天然的容错特性,因而本文利用事务存储系统的版本管理机制来进行故障恢复,提出一种基于事务回退的事务存储系统的故障恢复方法(FRTR).FRTR 在无故障时不会增加系统额外的检查点数据保存开销,发生故障时也只需对单事务进行回退,避免了无故障结点上的事务回退,从而减少了系统的故障恢复开销.本文称这种具有 FRTR 机制的事务存储系统为容错事务存储系统(fault tolerant transactional memory,简称 FTTM).

FRTR 机制中假设事务在执行 commit 指令之前存在有故障检测机制用以检测故障,当发现故障时系统即回退当前事务.为此,除支持传统事务系统中包括 begin,commit,abort 在内的事务指令外,FTTM 还需要引入一个 rollback 指令,用以显式地指示系统回退当前的故障事务.rollback 指令的执行语义是放弃故障事务对存储单元所做的修改,并将系统状态恢复至故障事务执行之前,重新执行该事务.图 1 对比了传统事务存储系统与引入了 rollback 指令的容错事务存储系统的编程模式.



(a) Traditional TM (b) FTTM

Fig.1 Programming model comparison between traditional and fault transactional memory

图 1 传统事务存储系统与容错事务存储系统编程模式对比

作为事务存储系统,容错事务存储系统中的事务也需要满足原子性(atomicity)和隔离性(isolation)[2,6−9].事务的原子性是指一个事务所做的工作要么全部提交要么全部丢弃,它确保了事务在出现故障时可以取消自己已有工作对系统产生的影响;而事务的隔离性又称为可串行性(serializability),是指事务的执行可以被视为一种串行执行,它保证了在故障事务提交前故障不会传播到其他的事务中.

显而易见,rollback 指令的引入并不会破坏事务存储系统中事务执行的原子性.但是由于 rollback 指令的执行可能会改变容错事务存储系统中事务原有的执行序列,所以存在破坏容错事务存储系统隔离性的可能.

可见,容错事务存储系统是否能够保证事务执行的隔离性是确定 FRTR 故障恢复方法正确性的关键.因此,本文第 2 节对容错事务存储系统的隔离性进行了深入的讨论,并证明 rollback 指令的引入不会破坏容错事务存储系统的隔离性,从而证明 FRTR 方法的正确性.

首先通过一个简单的例子来说明 rollback 指令对容错事务存储系统中事务执行序列所产生的影响.

1.2 一个例子

假设有容错事务存储系统内的两个事务 $T1$ 和 $T2$,在某一时刻分别对变量 x 进行读写操作.考虑 $T1$ 和 $T2$ 可能的执行情况,图 2 是引入了 rollback 指令的容错事务存储系统中的事务执行流程示意图.其中,图 2(a)和图 2(c)示意了无故障时 $T1$,$T2$ 的两种可能的指令相对执行序列,而图 2(b)和图 2(d)则分别示意了图 2(a)和图 2(c)所对应的在系统有故障时因执行 rollback 指令可能引起的指令相对执行序列的改变.

假设事务 $T2$ 在 $2t$ 时刻时执行事务提交操作,并假设事务在提交前进行故障检测.为了方便讨论,假设事务 $T1$ 和 $T2$ 的访问数据变量集合的交集中的元素只有变量 x,同时假设事务 $T1$ 和 $T2$ 对变量 x 写和读的操作指令分别有且只有一次,且 $T2$ 的读 x 操作指令的首次执行先于 $T1$ 的写 x 操作指令的首次执行,如图 2 所示,即 $T2$ 的指令 $C2$ 先于 $T1$ 的指令 $C1$ 执行.同时,假设系统采用的是 eager 的冲突检测机制,即对事务的每次读写操作都会进行冲突检测.下面考虑 $T1$,$T2$ 的两种可能的指令相对执行序列.

若 $T1$,$T2$ 的指令相对执行时序如图 2(a)所示,在系统无故障时,$T1$ 在执行 $C1$ 的 write x 操作时会因 $T2$ 中 $C2$ 的 read x 操作而发生冲突回退.在 $2t$ 时刻 $T2$ 完成正常提交,$T1$,$T2$ 的提交顺序应该是 $T2$ 先于 $T1$.

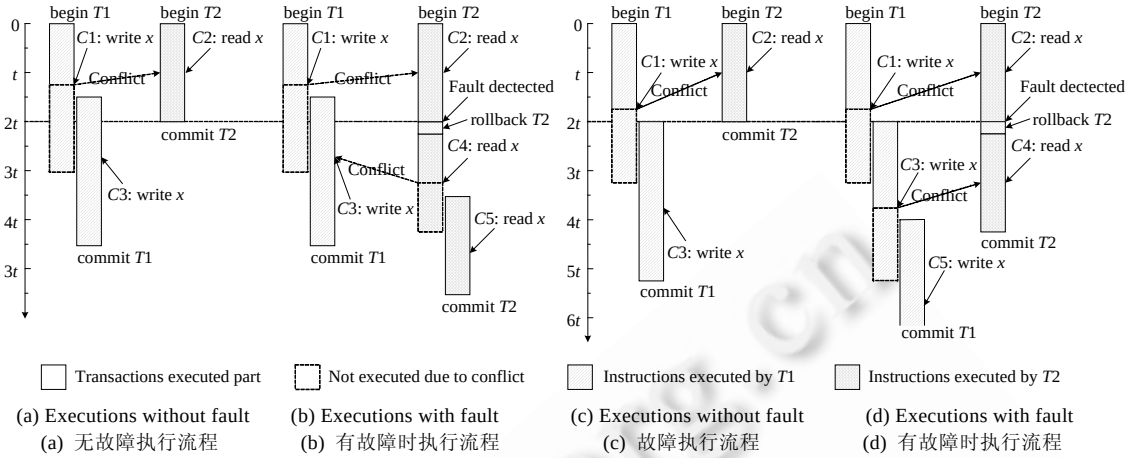
而若 $T2$ 在 $2t$ 时刻检测到故障并调用 rollback 指令进行故障恢复后重新执行该事务,如图 2(b)所示,则 $T2$ 在重新执行过程中执行 $C4$ 的 read x 的指令时,又会因与先前发生冲突回退的 $T1$ 执行的 $C3$ 的 write x 指令发生冲突而产生回退.这样,$T1$ 和 $T2$ 的提交顺序就变成了 $T1$ 先于 $T2$.

与图 2(a)类似,在图 2(c)中,当系统无故障发生时,$T2$ 会先于 $T1$ 完成提交.

而当事务 $T2$ 在 $2t$ 时刻检测到故障时,如图 2(d)所示,由于 $T1$ 的执行序列与图 2(a)和图 2(b)中 $T1$ 执行序列的区别,$T1$ 在执行 $C3$ 的 write x 操作时,会因与 $T2$ 调用 rollback 指令回退后所执行的 $C4$ 的 read x 操作发生冲突而致使事务 $T1$ 再次回退.虽然 $T1$ 和 $T2$ 的提交序列维持 $T2$ 先于 $T1$ 没有改变,但是注意到,$T1$ 因冲突而引发的回退由无故障时的一次变成了两次.也就是说,系统的指令执行序列发生了变化.

通过分析图 2 所假设的情况可以看出,rollback 指令的引入改变了系统的指令执行序列,并有可能使容错事务存储系统中发生故障和不发生故障时事务的提交顺序发生变化,这是因为单一事务的回退必然会导致故障

恢复前和故障恢复后系统状态的不一致.不同于经典的 rollback-recovery 机制是对整个系统状态的恢复,FRTR 方法的故障恢复并不保证整个系统恢复到一个与故障前完全相同的系统状态.



(a) Executions without fault (b) Executions with fault (c) Executions without fault (d) Executions with fault

(a) 无故障执行流程 (b) 有故障时执行流程 (c) 故障执行流程 (d) 有故障时执行流程

Fig.2 Transactions execution in FTTM

图 2 容错事务存储系统中的事务执行流程示意图

虽然在并发事务系统中这种不确定的事务提交顺序是合法的,但是,这种因 rollback 指令的引入而造成的系统状态的改变,对于容错事务存储系统的隔离性是否会造成影响需要进行进一步分析和讨论.本文第 2 节将重点对容错事务存储系统的隔离性进行讨论,并以此证明 FRTR 方法能够保证容错事务存储系统中故障的正确恢复.

1.3 FRTR开销分析

第 1.1 节中提到 FRTR 方法在故障恢复时所使用的恢复数据是事务执行时本身的数据版本管理机制所保存的“检查点”,所以如果不考虑检错开销,在无故障情况下,FRTR 方法几乎不会增加额外的开销.而在系统检测到故障时所增加的故障恢复开销,也主要是单个事务的回退开销和故障事务重新执行的执行开销.

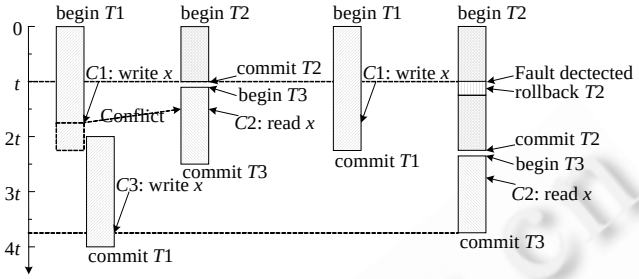
对于故障事务重新执行的执行开销,一方面,在传统的后向故障恢复过程中这是不可避免的固定开销;另一方面,由于事务的粒度一般相对较小,所以 FRTR 方法所增加的执行开销也相对较小.

而单个事务的回退开销,依据系统所采用的数据版本管理机制而有所不同.对于采用基于缓存的数据版本管理机制的系统,事务回退的过程只是清空缓存,开销几乎可以忽略;对于采用基于日志的数据版本管理机制的系统,事务回退的过程是依据日志的索引恢复故障事务所修改的变量的旧值,开销也相对较小.

另外,本文在对图 2 的分析中提到,FRTR 方法使故障恢复后与故障恢复前的系统状态发生了变化;并且在图 2 中也可以看到,由于容错事务存储系统本身的冲突检测机制的存在,使得 FRTR 方法的故障恢复开销不等于单个故障事务的回退开销与故障事务重新执行一次的执行开销之和.如图 2(b)所示,FRTR 方法的故障事务 $T2$ 的重新执行的执行开销就因为指令 $C3$ 和 $C4$ 所引起的冲突而大于故障事务 $T2$ 执行一次的开销.而在图 2(d)中,由于 $T2$ 的故障恢复引起的指令 $C3$ 和 $C4$ 的冲突使得 $T1$ 的再次回退,也造成了程序整体执行时间的增加.但是,这并不是说 FRTR 方法的故障恢复开销一定大于单个故障事务的回退开销与故障事务重新执行一次的执行开销之和.

考虑一个稍微复杂点的情况,如图 3 所示,假设有 3 个事务 $T1$,$T2$,$T3$:$T1$ 在处理器核 $P1$ 上运行,$T2$ 和 $T3$ 在处理器核 $P2$ 上运行,并假设 $T1$ 和 $T3$ 分别有且只有一次对变量 x 写和读的操作指令,其他假设与图 2 类似.由图 3(a)可以看到,0 时刻,$T1$ 和 $T2$ 两个事务同时在两个处理器核上开始执行,$T2$ 在 t 时刻提交事务;随后,事务 $T3$ 在该核上继续执行.而 $T1$ 在 $1.75t$ 时刻执行指令 $C1$ 的 write x 操作时,与 $T3$ 执行的指令 $C2$ 的 read x 发生冲突,产生回退.这样,$P1$ 上的任务 $4t$ 时刻完成,而 $P2$ 上的任务在 $2.5t$ 时刻完成,整个系统在 $4t$ 时刻完成全部任务.

而在图 3(b)中,假设 $T2$ 在 t 时刻检测到故障进行回退,假设 $T2$ 的回退开销为 $0.25t$.可以看到,由于 $T2$ 的故障回退,推迟了 $T3$ 的执行,使得在 $1.75t$ 时刻 $T1$ 执行指令 $C1$ 的 write x 操作时没有因冲突而回退,最终,$P1$ 在 $2.25t$ 时刻就完成了任务,而 $P2$ 由于 $T2$ 的故障回退完成任务的时间推迟到 $3.75t$ 时刻,但是整个系统的完成任务时间非但没有增加反而提前了 $0.25t$ 时间.



(a) Executions without fault (b) Executions with fault

(a) 故障执行流程 (b) 有故障时执行流程

Fig.3 Overhead analysis diagram of FRTR

图 3 FRTR 开销分析示意图

从图 3 可以看到,采用 FRTR 方法进行故障恢复,对整个系统而言所引入的故障恢复开销是比较复杂的,这主要是源于 FRTR 方法的故障恢复并不保证整个系统恢复到一个与故障前完全相同的系统状态,以至于系统的事务执行序列发生变化.但总体来讲,在检测到故障时,FRTR 方法所增加的故障恢复开销相比于经典的 rollback-recovery 技术还是很小的,这一点本文第 3 节的实验结果可以给予很好的证明.

2 容错事务存储系统的隔离性

在有些经典的 rollback-recovery 技术中,由于检查点是独立保存的,不能保证故障被很好的隔离,以至于在故障恢复时可能出现回退后各结点不一致的系统视图,致使系统不断地回退,直至程序的初始状态,产生所谓的“多米诺效应”[5].在容错事务存储系统中,事务的“检查点”也是各结点独立的,而且由第 1.2 节的例子可以看到,rollback 指令的引入有可能造成容错事务存储系统中在发生故障和不发生故障时事务提交顺序的变化.因此,要保证单事务回退的 FRTR 机制可以有效地进行故障恢复,就需要证明容错事务存储系统中事务执行的隔离性.显而易见,在单结点系统中,由于事务是串行执行的,所以任意一个事务的回退都不存在系统状态不一致的问题.因此,如果在并发容错事务存储系统中事务的并发执行可以等价成事务的某种串行执行序列,则可以认为任意一个事务的回退都不会引起不一致的系统视图.

定义 1(隔离性[2,6−8]). 在容错事务存储系统中,事务虽然可以并发地执行,但其执行结果与某种顺序执行的结果一样,即系统行为等价于事务的某种串行执行,有一种一个事务执行完毕后,下一个事务才开始执行的表象.本文称这样的容错事务存储系统是满足隔离性的.

显然,如果容错事务存储系统具备隔离性,则确保了在故障事务提交前故障不会传播到系统内其他事务.因此,FRTR 方法的单一事务的回退可以正确地完成系统的故障恢复,而不会引起不一致的系统视图.

为了对容错事务存储系统的隔离性进行讨论,本文首先对容错事务存储系统进行必要的抽象.

在传统的事务存储系统中,一个事务序列是以 begin 指令开头,以 commit 或者 abort 指令作为结尾,其间包含包括分支、循环、过程调用等程序流控制指令在内的指令序列.容错事务存储系统在此基础上引入了一个 rollback 指令,用以显式指示故障事务的回退.显然,会对事务系统隔离性产生影响的只有那些具有对外部操作的程序行为指令.因此,为简化讨论,本文这里只考虑包括 commit,abort,rollback,read 和 write 在内的有对外部操作的程序行为指令序列.

令 $V=\{v_1,v_2,\ldots,v_k\}$,表示容错事务存储系统中所涉及的 k 个变量的集合.

令 A={read,write},表示读、写的指令集合.

令 C={commit,abort,rollback}∪{$A\times V$},表示操作集合 V 中变量的指令集合.

令 $T=\{T_1,T_2,\ldots,T_n\}$,表示容错事务存储系统中的事务集合.

可以将事务的执行序列表示成如下的二元组

$$\langle\langle t,c_i\rangle|i=1,2,\ldots,n\rangle,c_i\in C,t\in T.$$

其含义是事务 t 的第 i 步执行的指令 c_i.

因此,可以用$\{\langle t,c\rangle|c\in C\}$表示事务 t 的执行序列集合.

本文用 $R(t)$表示事务 t 的读集合,$W(t)$表示事务 t 的写集合,则

$$R(t)=\{v|\langle t,\langle \text{read},v\rangle\rangle\in t\}.$$

同理,

$$W(t)=\{v|\langle t,\langle \text{write},v\rangle\rangle\in t\}.$$

定义 2(冲突). 在容错事务系统中,事务冲突可定义为:若两个执行中的事务 T_i 和 $T_j(i\neq j)$存在冲突,则有

$$R(T_i)\cap W(T_j)\cup W(T_i)\cap R(T_j)\cup W(T_i)\cap W(T_j)\neq\varnothing.$$

定义 3(调度). 容错事务存储系统的所有操作,在保持其在事务内各自顺序的前提下,可以任意合并成一个序列,本文称之为容错事务系统的一次调度,表示为

$$S=\langle\langle t,c\rangle_i|i=1,2,\ldots,n\rangle,t\in T.$$

由定义 3 可知,调度中的一步是事务 t 执行的一个指令 c,本文用 $S[i]$表示调度 S 的第 i 步操作.

容错事务存储系统事务集 T 的一次调度反映了系统的一次指令执行的时间序关系.对于串行系统,这个序关系就是指令执行的先后顺序;而对于并发系统,可以设想存在一个全局的足够精确的时钟,每个结点的每条指令操作在完成时都可以通过时钟得到一个时间戳,这些时间戳为所有的指令操作生成了一个时间序.显然,事务集 T 的一次调度集是一个全序集合.

显然,串行系统的任意一次调度 S 应满足如下条件:

对于$\forall T_i,T_j\in T$,或者 T_i 的所有指令操作都先于 T_j 的第 1 条指令操作,或者 T_j 的所有指令操作都先于 T_i 的第一条指令操作.

本文称这样的调度 S 为串行调度,并用 S_{seq} 表示.显然,串行调度是满足隔离性的.

在并发系统中,存在着一个事务读或者写另外一个事务的“脏数据”的可能性,这种一个事务对另一个事务的“脏数据”进行操作称之为事务的相关关系.显然,这种相关关系是影响系统隔离性的原因.比较容错事务存储系统中冲突的定义,可以将容错事务系统中事务冲突理解为两个执行中的事务存在着相关关系.

对于 T 的每次调度 S,定义一个三元相关关系集合 $Dep(S)$,其定义如下:

假设 T_1 和 T_2 是 T 中任意两个不同事务,v 是任意一个变量,并设 $i,j(i<j)$是 S 中的任意两个操作.假设 $S[i]$涉及 T_1 在 v 上的操作 a_1,$S[j]$涉及 T_2 在 v 上的操作 a_2,并假设在 a_1 和 a_2 之间没有任何事务对 v 的写操作(即在 $S[i+1],\ldots,S[j-1]$中没有$\langle T',\langle \text{write},v\rangle\rangle$,$T'$为 T 中的任意事务).于是,$Dep(S)$可以定义为

$$\langle T_1,v,T_2\rangle\in Dep(S),\ \text{if } a_1=\text{write or } a_2=\text{write}.$$

对于 T 上的两次调度 S 和 S',如果满足 $Dep(S)=Dep(S')$,则称调度 S 与调度 S'是等价的,记为 $S\equiv S'$.

定义 4(调度的隔离性). 如果 T 上的一次调度 S 满足 $S\equiv S_{seq}$,那么调度 S 是满足隔离性的.

由 $Dep(S)$的定义可以看到,容错事务存储系统的一次调度 S 中的相关关系定义了事务间的一种时序约束,本文用符号$<_s$来标记调度 S 上的这种时序关系.

对于 $T_1,T_2\in T$,若 $T_1<_sT_2$,当且仅当$\langle T_1,v,T_2\rangle\in Dep(S)$,其中,$v\in V$.

时序$<_s$定义了事务执行的一种时序关系,即若 T_i,T_j 满足 $T_i<_sT_j$,则表示 T_j 读写了 T_i 写过的数据,或是 T_j 重写了 T_i 读过的数据.

本文将容错事务存储系统的一次调度 S 的相关关系定义成一个有向图 $G=\langle W,E\rangle$,其中,

$$W=T,E=\{\langle T_i,T_j\rangle|\langle T_i,v,T_j\rangle\in Dep(S),T_i,T_j\in T,i\neq j,v\in V\}.$$

定理 1(隔离性定理). 容错事务存储系统的事务集 T 的一次调度 S 是满足隔离性的,当且仅当 S 的相关关系图 G 是无环图.

证明:首先,本文采用反证法证明隔离性⇒无环图.

因为 S 是事务集 T 上的一次满足隔离性的调度,根据定义 4,必然存在一个串行调度 S_{seq},满足 $S\equiv S_{seq}$,不妨假设 $S_{seq}=\langle T_i|i=1,2,\ldots,n\rangle$.假设 S 的相关关系图 G 有环,则存在一个事务序列$\langle T_j,T_{j+1},\ldots,T_k\rangle(j<k)$满足 $T_i<_sT_{i+1}(i=j,j+1,\ldots,k-1)$且有 $T_k<_sT_j$.因为 $j<k$,所以在串行调度 S_{seq} 中只有 $T_j<_{seq}T_k$,即在调度 S_{seq} 中不存在 $T_k<_{seq}T_j$.又因为 $S\equiv S_{seq}$,根据等价调度的定义,有 $Dep(S)=Dep(S_{seq})$,即在调度 S 中也不存在 $T_k<_sT_j$,与假设中 $T_k<_sT_j$ 矛盾.因此假设不成立,即如果调度 S 满足隔离性,则 S 的相关关系图 G 是无环图.

接下来,本文将采用数学归纳法来证明无环图⇒隔离性.

假设对于任意有 k 个事务的事务集 T,若其上的一次调度 S 的相关关系图 G 是无环图,则调度 S 是隔离的.

显然,当 $k<2$ 时归纳假设成立.

假设当 $k<n-1$ 时归纳假设成立,考虑 $k=n$ 时调度 S 的隔离性.

取事务 $t\in T$ 和 $t'\in T$ 满足 $t<_st'$,再取事务 $t''\in T$ 满足 $t'<_st''$,…,以此类推,可以得到一个序列 $S_t=\langle t,t',t'',\ldots\rangle$.若 S_t 是无限的,则必然会存在一个事务 $t_i\in T$ 出现两次,即存在一个序列$\langle t_i,t_{i+1},\ldots,t_j\rangle$满足 $t_m<_st_{m+1}(m=i,i+1,\ldots,j-1)$且有 $t_j<_st_i$,与 S 的相关关系图是无环图矛盾,所以 S_t 必是有限的.不妨假设 S_t 中的最后一个事务是 t,则调度 S 中不存在事务 $t\in T$ 满足 $t<_st$.

考虑调度 $S'=\langle\langle t_i,a_i\rangle\in S|t_i\neq t^*\rangle$,有 $Dep(S')=\{\langle t',v,t''\rangle\in Dep(S)|t'\in T,t''\in T,t''\neq t^*,v\in V\}$.

因为 S 的相关关系图是无环图,所以 S'的相关关系图也必是无环图,且 S'中只包含 $n-1$ 个事务.根据归纳假设 S'是隔离的.不妨假设串行调度 $S'_{seq}\equiv S'$,并假设 $S'_{seq}=\langle t_i\,|\,i=1,2,...,n-1\rangle$.

构造串行调度 $S_{seq}=S'_{seq}\,\|\,t^*$,且

$$Dep(S_{seq})=Dep(S'_{seq}\,\|\,t^*)=Dep(S'_{seq})\cup\{\langle t',v,t^*\rangle\in Dep(S)\,|\,t'\in T,v\in V\}.$$

因为 $S'_{seq}\equiv S'$,所以 $Dep(S'_{seq})=Dep(S')$,将 $Dep(S')$的等式代入上述等式,有

$$Dep(S_{seq})=Dep(S')\cup\{\langle t',v,t^*\rangle\in Dep(S)|t'\in T,v\in V\}=Dep(S),$$

即 $S_{seq}\equiv S$,所以 S 是隔离的得证.

综上,定理 1 得证. □

容错事务存储系统按照冲突检测机制可以分为 eager 和 lazy 两类[1,2],若基于 eager 和 lazy 两种冲突检测机制下的容错事务系统的合法调度都是满足隔离性的,那么容错事务系统的任何合法调度都是满足隔离性的.为证明容错事务存储系统的隔离性,本文将分别针对 eager 和 lazy 两种冲突检测机制对容错事务存储系统的隔离性进行讨论.

2.1 基于eager冲突检测机制的容错事务存储系统的隔离性

eager 冲突检测机制是在事务存储系统的每次读或者写操作都进行相关性检测,一旦发现冲突即回退冲突的事务.基于 eager 冲突检测机制的事务存储系统根据采用的数据版本管理机制又可以分为基于 eager 数据版本管理(如 logTM[10])和基于 lazy 数据版本管理(如 LTM[11])两种.前者直接将新值写入目标地址并维护读写日志;后者缓存新值,直到 commit 指令执行.

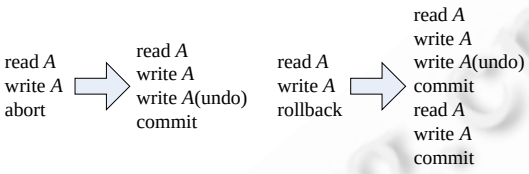
由于冲突检测的动作是在执行每一次读写指令时发生的,而 commit 指令的执行并不会引发冲突,所以 commit 指令可以认为只是对事务 t 的 $R(t)$和 $W(t)$集合的释放.尽管在 lazy 的版本管理机制下,对数据真正的写实际上是发生在执行 commit 指令时,但是由于执行 commit 指令时出现的写操作并不会引起冲突发生,对系统的隔离性不产生影响,因此可以忽略 lazy 的版本管理机制下的 commit 的写动作.

为方便讨论,在不改变指令语义的前提下,本文对 eager 冲突检测机制下的容错事务存储系统的 abort 和 rollback 指令做以等价映射.

指令 abort 是对当前事务所做写操作的废弃,可以等价地认为 abort 指令执行了对 $W(t)$集合内的元素执行了

一系列特殊的写指令——undo 写指令之后,执行 commit 指令,释放事务的读写集合.虽然在 lazy 的版本管理机制下,abort 指令实际上是进行了清空了事务的读写 buffer 的动作,并不会带来额外的写操作,但是这种等价并不会丢失对破坏隔离性情况的判断.如图 4(a)所示,左边代码中的 abort 指令可以等价成右边的 write *A*(undo)和 commit 两条指令序列.

同理,容错指令 rollback 所做的操作实际是废弃当前事务 *t* 的写操作后,重新执行该事务.可以将一个执行了 rollback 指令的事务等价成一个执行了 abort 指令的事务 *t* 和一个新的事务 *t*′的串行执行.显然,*t* 与 *t*′是严格串行的,且 *t*′具有和 *t* 相同的读写指令序列.如图 4(b)中所示,左边代码中的 rollback 指令可以等价成右边的第 3 条~第 7 条指令的指令序列.



(a) (b)

Fig.4 Equivalent mapping of abort and rollback based on eager conflict detection mechanism

图 4 eager 冲突检测机制下,abort,rollback 指令的等价映射

经过上面的指令等价映射,基于 eager 冲突检测机制的容错事务存储系统的指令集合可以简化如下:

$$C_{eager}=\{\text{commit}\}\cup A\times V.$$

这样,基于 eager 冲突检测机制的容错事务存储系统中的一个合法的事务只包含 commit,read,write 等 3 种指令,其中,commit 指令有且仅有一次,且该 commit 指令的执行序在该事务的所有 read 和 write 指令之后.

根据定义 2,可以对基于 eager 冲突检测机制的容错事务存储系统的调度的合法性进行定义:

1) 如果一个事务 *t* 对变量 *v* 执行了操作⟨read,*v*⟩,则在 *t* 执行 commit 操作之前,任何事务的⟨write,*v*⟩操作都是不合法的;
2) 如果一个事务 *t* 对变量 *v* 执行了操作⟨write,*v*⟩,则在 *t* 执行 commit 操作之前,任何事务的⟨*a*,*v*⟩(*a*∈*A*)操作都是不合法的.

对于一个合法的事务集 *T*,为 *T* 上的每个事务 *t* 定义 *Comm*(*t*),用以表示在 *T* 上的一个合法调度 *S* 中 *t* 的 commit 指令的索引,即 *Comm*(*t*)=*i*,其中,*i* 满足 *S*[*i*]=⟨*t*,commit⟩.

因为 *T* 中的每个事务都是合法的,即 *T* 中的每个事务都包含且仅包含一条 commit 指令,所以每个事务都可以定义出这样一个唯一的 *Comm* 值.

引理 1. 对事务集 *T* 中的两个事务 *t* 和 *t*′,如果 $t<_s t'$,则有 *Comm*(*t*)<*Comm*(*t*′).

证明:假设 *S* 为事务集 *T* 上的一次合法调度,*S* 中的两个步骤 *i*,*j* 满足 *i*<*j*,*S*[*i*],*S*[*j*]分别是 *S* 中事务 *t* 和 *t*′关于变量 *v* 的操作,即 *S*[*i*]=⟨*t*,⟨*a*,*v*⟩⟩,*S*[*j*]=⟨*t*′,⟨*a*′,*v*⟩⟩.因为 $t<_s t'$,由 *Dep*(*S*)的定义,*a* 和 *a*′中至少有一个是 write 指令.

因为 *S*[*i*]=⟨*t*,⟨*a*,*v*⟩⟩是 *T* 上的一次合法调度,由调度合法性的定义可知,必然存在 *S* 中的步骤 *k* 满足 *k*<*j*,且 *S*[*k*]=⟨*t*,commit⟩.另一方面,作为合法事务,*t*′的 commit 指令执行必在 *S*[*j*]之后,即 *j*<*Comm*(*t*′).

综上,有 *Comm*(*t*)=*k*<*j*<*Comm*(*t*′),引理 1 得证. □

定理 2. 在 eager 冲突检测机制下,容错事务存储系统的任何一次合法调度都是隔离的.

证明:本文采用反证法来证明定理 2.假设事务集 *T* 上存在一个合法的调度 *S*,且 *S* 不是隔离的.

根据定理 1,必然会存在一个事务序列⟨$T_1,T_2,\ldots,T_n$⟩满足 $T_i<_s T_{i+1}$(*i*=1,2,…,*n*−1),且 $T_n<_s T_1$.根据引理 1,有 *Comm*(T_1)<*Comm*(T_2)<…<*Comm*(T_n)<*Comm*(T_1).显然,*Comm*(T_1)<*Comm*(T_1)是矛盾的,所以假设不成立.即不存在这样一次不是隔离的合法调度 *S*,即在 eager 冲突检测机制下,容错事务存储系统的任何一次合法调度都是隔离的.定理 2 得证. □

定理 2 证明了 eager 冲突检测机制下容错事务存储系统的隔离性,本文下面将讨论 lazy 冲突检测机制下的

容错事务存储系统的隔离性.

2.2 基于lazy冲突检测机制的容错事务存储系统的隔离性

不同于 eager 的冲突检测机制,lazy 的冲突检测机制是在事务执行 commit 指令时才进行相关性检测.基于 lazy 冲突检测机制的容错事务存储系统采用的都是 lazy 的版本管理方式(如 TCC[12]).也就是说,事务 t 对于某个数据变量 v 的第 1 次操作时会在本地数据缓冲中形成一个副本 v_t,之后对 v 的所有操作都由对 v_t 的操作取代,直到事务执行 commit 指令.

在基于 lazy 冲突检测机制的容错事务存储系统中,事务 t 执行 commit 指令时,所有被 t 执行过写操作的数据副本会取代存储器中相应的数据变量.即,若 $v\in W(t)$,则在 t 执行 commit 指令时将副本 v_t 的值写入 v 的地址.也就是说,事务 t 对其 $W(t)$集合的数据的写操作是在 commit 指令执行时才被真正执行.

同样的,类似于在 eager 冲突检测机制下对容错事务存储系统隔离性的讨论,在不改变指令语义的前提下,本文对 lazy 冲突检测机制下的容错事务存储系统的 abort 和 rollback 指令作以等价映射.

前面提到,指令 abort 是对当前事务所做写操作的废弃,而在基于 lazy 冲突检测机制的容错事务存储系统中,事务执行过程中写操作的对象是本地的数据副本,在 commit 指令执行之前,对于原数据变量而言可以认为这些写操作并没有实际发生.由于对系统不会产生影响,在后面的讨论中可以忽略执行了 abort 操作指令的事务.即对于执行了 abort 操作指令的事务 t,在事务集 T 的相关关系图 G 中不考虑关于 t 的结点和边.

同理,由于执行了 rollback 指令的事务可以等价成一个执行了 abort 指令的事务 t 和一个新事务 t'的串行执行,而事务 t 实际上是可以被忽略的,所以对于执行了 rollback 指令的事务可以直接等价成一个新的事务 t'.

经过上面的指令等价映射,基于 lazy 冲突检测机制的容错事务存储系统的指令集合也可以简化为

$$C_{lazy}=\{\text{commit}\}\cup A\times V.$$

这样,基于 lazy 冲突检测机制的容错事务存储系统的一个合法的事务也只包含 commit,read,write 这 3 种指令,而其中 commit 指令有且仅有一次,且该 commit 指令的执行序在该事务的所有 read 和 write 指令之后.

由于基于 lazy 冲突检测机制的容错事务存储系统中事务的写操作是对数据副本进行的,冲突的发生实际是在 commit 指令执行时出现的,根据冲突的定义,对基于 lazy 冲突检测机制的容错事务存储系统的调度的合法性可以给出如下定义:

对于$\forall T_m,T_n\in T(m\neq n)$,假设对于调度 S 中步骤 i,j 有 $S[i]=\langle T_m,\text{commit}\rangle,S[j]=\langle T_n,\text{commit}\rangle$,且 $p=\min\{x|S[x]=\langle T_n,\langle a,v\rangle\rangle,a\in A\}$,若 $v\in W(T_m)$,假设 $q=\min\{x|S[x]=\langle T_1,\langle\text{write},v\rangle\rangle\}$,则有 $i>q>j$ 为真或者 $i<p$ 为真.

这个定义表明,在基于 lazy 冲突检测机制的容错事务存储系统的一次合法的调度中,一个事务执行 commit 操作时,所有与其存在冲突的事务要么已经执行完 commit 操作,要么引发冲突的操作还没有开始执行.

定理 3. 在 lazy 冲突检测机制下,容错事务存储系统的任何一次合法调度都是隔离的.

证明:反证法.假设事务集 T 上存在一个合法调度 S,且 S 不是隔离的,根据定理 1,存在一个事务序列 $\langle T_1,T_2,\ldots,T_n\rangle$满足 $T_i<_sT_{i+1}(i=1,2,\ldots,n-1)$,且 $T_n<_sT_1$.

同样,为事务集 T 上的每个事务 t 定义 $Comm(t)$,用以表示在 T 上的一个合法调度 S 中 t 的 commit 指令的索引.不妨假设 $Comm(T_n)=\max\{x|x=Comm(T_i),i=1,2,\ldots,n\}$,则显然有 $Comm(T_1)<Comm(T_n)$.

由假设可知 $T_n<_sT_1$,根据时序关系$<_s$的定义,$\exists j,k$ 为调度 S 中的两个步骤,满足 $j<k$,且$\exists v\in V$,有 $S[j]=\langle T_n,\langle a_n,v\rangle\rangle$ 和 $S[k]=\langle T_1,\langle a_1,v\rangle\rangle$,其中 $a_1,a_n,\in A$,且 a_1,a_n 中至少有一个为 write 操作.

不妨假设 a_n 为 write 操作,即 $v\in W(T_n)$.令 $p=\min\{x|S[x]=\langle T_1,\langle a,v\rangle\rangle,a\in A\}$,则有 $p<k$.因为 T_1 是合法事务,所以有 $k<Comm(T_1)$.又因为 $Comm(T_1)<Comm(T_n)$,所以有 $p<Comm(T_n)$.

再令 $q=\min\{x|S[x]=\langle T_n,\langle\text{write},v\rangle\rangle\}$,则有 $q<j$.又因为 $j<k$,所以有 $q<Comm(T_1)$.

而另一方面,已知 S 是事务集 T 上的一次合法调度,根据调度合法性的定义可知,应有 $Comm(T_n)<p$ 为真或 $Comm(T_n)>q>Comm(T_1)$为真.显然与 $p<Comm(T_n)$且 $q<Comm(T_1)$的结论矛盾.所以,a_n 为 write 操作的假设不成立.

根据假设 a_1,a_n 中至少有一个为 write 操作,而 a_n 不可能是 write 操作,所以必有 a_1 为 write 操作,即 $v\in W(T_1)$.因为 T_1 是合法事务,所以有 $k<Comm(T_1)$.令 $m=\min\{x|S[x]=\langle T_n,\langle a,v\rangle\rangle,a\in A\}$,则有 $m<j$.

又因为 $j<k$ 且 $Comm(T_1)<Comm(T_n)$,所以有 $m<Comm(T_1)<Comm(T_n)$.

同样,因为 S 是事务集 T 上的一次合法调度,根据调度合法性的定义可知,应有 $Comm(T_1)<m$ 为真或者 $Comm(T_1)>Comm(T_n)$为真,与 $m<Comm(T_1)<Comm(T_n)$矛盾,因此 a_1 也不可能为 write 操作.

综上所述,存在这样一次不是隔离的合法调度 S 的假设不成立,即在 lazy 冲突检测机制下,容错事务存储系统的任何一次合法调度都是隔离的,定理 3 得证. □

2.3 FRTR机制对故障恢复的充分性

经过对 eager 和 lazy 两类冲突检测机制下的容错事务存储系统的隔离性的讨论,可以得出关于容错事务存储系统隔离性的结论.

定理 4. 容错事务存储系统的任何一次合法调度都是隔离的.

证明:容错事务存储系统按照冲突检测机制可以分为 eager 和 lazy 两类,通过定理 2 和定理 3 的证明可知,在这两类冲突检测机制下的容错事务存储系统的任何一次合法调度都是隔离的.因此,容错事务存储系统的任何一次合法调度都是隔离的,得证. □

本文假设了容错事务存储系统的存储系统是可靠的,并假设所有可能出现的瞬时故障都是发生在计算过程中,所以故障的传播仅是通过对数据的写操作在事务间进行的.定理 4 证明了容错事务存储系统的任何一次合法调度的隔离性,也就是说,容错事务存储系统的任何一次合法调度都可以等价成一个串行调度.这样,任何一个在事务执行过程中出现的瞬时故障在事务执行 commit 指令操作之前都是不会被传播至其他事务的.因此,在事务执行 commit 指令操作之前进行的故障检测所检测到的故障的影响范围仅限于当前的故障事务.

容错事务存储系统的原子性确保了故障事务所进行的写操作的可回退性,而其隔离性又保证了故障不被传播.因此,发生故障时采取单事务回退的 FRTR 故障恢复机制对于容错事务存储系统的故障恢复是充分的.

3 实 验

3.1 实验方法

通过对容错事务存储系统的隔离性的讨论,本文证明了 FRTR 方法对于事务存储系统的故障恢复的正确性和充分性,本节选取了 5 个测试用例进行实验对 FRTR 方法的性能进行验证.

本文采用的 5 个测试用例中,包括 4 个 SPLASH-2 中的测试用例 barnes,lu,ocean 和 cholesky 和 gems2.1 测试用例包中的一个事务测试用例 deque.SPLASH-2 是由 Stanford 开发的一套针对共享存储的并行应用的标准测试用例[4].

- Barnes 是一个模拟物体间相互作用关系的一个应用程序,其输入参数是物体数.
- Lu 是一个将稠密矩阵分解成一个上三角矩阵和一个下三角矩阵的 LU 分解算法的核心代码段,其输入参数是矩阵规模.
- Ocean 是一个研究涡流和边界流对于大范围海洋运动的影响的应用程序,其输入参数是海洋被划分的网格数.
- Cholesky 是一个将稀疏矩阵分解成一个下三角矩阵和其转置的 Cholesky 分解算法的核心代码段,其输入为保存稀疏矩阵的文件.
- Deque 是一个同时向一个队列的头尾增减数据的测试用例,其输入参数就是增减数据的次数.

由于 SPLASH-2 标准测试本身不是针对事务存储的,所以本文将 barnes,lu,ocean 和 cholesky 这 4 个典型用例的主要循环程序段以循环迭代为单位进行事务化.而测试用例 deque 取自 gems2.1 的 transactional microbenchmarks,本身就是一个事务程序.表 1 是这些测试用例的基本信息.

Table 1 Test Cases

表 1 测试用例

	Scale parameter	Transactions number	Fault transactions number	Checkpoint size (B)
Barnes	16 384	131 072	1 312	34 390 824
Lu	512	15 248	153	2 113 736
Ocean	258	18 432	186	14 911 048
Cholesky	Lshp.o	11 368	114	1 310 720
	D750.o	1 504	16	8 363 240
Deque	360 000	360 000	3 600	99 646

由于本文只是要验证 FRTR 故障恢复方法的性能,为排除故障检测机制对性能的影响,本文并没有进行实际的故障注入实验,而是在程序运行过程中随机对程序中 1%的事务进行故障标注,并采用 FRTR 方法对被标注故障的事务进行故障恢复.

另一方面,作为对比,本文在系统中模拟了近似的系统检查点机制.因为容错事务存储系统 FRTR 机制的回退粒度是以事务为单位的,所以本文中作为对比的系统检查点机制的检查点保存时机也就设置在每个事务 begin 操作指令之前进行,同样随机对程序中 1%的事务进行故障标注,对于出现故障的事务系统恢复到最近的检查点重新执行.

实验中,程序运行在 simics3.0.29+gems2.1 模拟的四核 logTM 系统,每个处理器核的主频为 75MHz,系统主存 2GB,磁盘 8GB,系统所运行的操作系统为 solaris9.

3.2 实验结果

本节对 FRTR 机制的性能进行了评估,并与系统级 checkpoint&restart(C&R)技术进行了性能对比.本文针对每个测试用例测试以下 5 组数据:

- 原始事务程序的运行时间.
- FRTR 机制在无故障情况下的运行时间.
- FRTR 机制在有故障情况下的运行时间.
- C&R 机制在无故障情况下的运行时间.
- C&R 机制在有故障情况下的运行时间.

图 5 为各个测试用例的这 5 组实验结果数据.通过对各项数据的观察,可以得到如下结论:

(1) 原始事务程序的运行时间与 FRTR 机制在无故障情况下的运行时间几乎相同.这是由于 FRTR 机制在故障恢复所使用的恢复数据是事务执行时本身的数据版本管理机制所保存的“检查点”,从而避免了额外的检查点保存开销,使得在无故障情况下系统几乎不需要为可能发生的故障恢复进行额外的操作.

(2) FRTR 机制在有故障时所增加的故障恢复开销是很小的.第 1.3 节中对 FRTR 方法的恢复开销做了定性的分析,说明了 FRTR 的故障恢复开销主要包括故障事务的回退开销和重新执行故障事务的开销.由于 logTM 的可用于故障恢复的日志文件是存放在高速存储器中,相比于存储在稳定存储器的检查点,FRTR 的单个故障事务的回退开销是相对较小的.而重新执行故障事务的开销,由于实验中仅对 1%的事务进行了故障标注,所以对于整个程序而言,重新执行故障事务的开销只相当于多执行了 1%的事务.考虑到第 1.3 节提到的 FRTR 方法在进行故障恢复时给系统带来的复杂扰动,相比于无故障运行,增加的开销主要取决于故障事务占整个程序的比重以及由于故障恢复所带来的冲突回退的数量.Gems2.1 的事务测试用例 deque 因其是一个全事务化用例,即故障事务占整个程序的比重较大,且 deque 用例在运行过程中冲突发生比较频繁,故障回退相对会带来更多的冲突回退,因此故障恢复开销最大,但也不足 7.7%.而选用 SPLASH-2 的 4 个用例由于不是全事务化用例而且事务间的冲突没有 deque 那样剧烈,所以恢复开销更是低至 0.11%~1.1%不等.

(3) 在 C&R 机制中,由于硬件事务存储系统中事务的粒度相对较小,而检查点要保存的数据量相对较大,所以相比于计算,I/O 的延时变得比较突出,保存检查点的开销就成为了程序执行的主要开销.可以看到:barnes 由于检查点文件较大且需要保存检查点的次数较多,所以检查点的保存开销最大,无故障时的运行开销相比于原

始事务程序增加了 15 000 多倍;而 cholesky 在 d750 规模下由于需要保存检查点的次数相对较少,所以无故障时的运行开销相比于原始事务程序的增加量最少,但也增加了 360 多倍.相对地,用例 cholesky 在 lshp 规模下由于保存次数较 d750 规模增加很多,开销也增加了 2 600 多倍;而 deque 尽管保存次数很多,但由于保存数据量很小,因此开销增加量相对较小,在 1 800 倍左右.

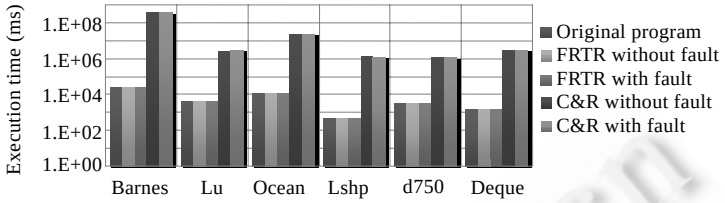


Fig.5 Comparison of execution time

图 5 执行时间比较

为了更清楚地体现 FRTR 机制的故障恢复开销的优势,图 6 仅就 FRTR 和 C&R 两种机制的故障恢复时间进行了比较.可以看到,C&R 的故障恢复时间较于 FRTR 的故障恢复时间,除了 deque 用例由于故障回退引起的系统扰动比较明显,使得 C&R 的故障恢复时间只比 FRTR 的故障恢复时间增加了 60 倍左右,其他用例增加量在 128 倍~4164 倍不等.究其原因,C&R 的故障恢复和 FRTR 的故障恢复方法都引入了故障事务重新执行的时间开销,所以主要的性能差别就在于系统状态回退的开销.而一方面,相比于存储在稳定存储器的检查点的 I/O 操作开销,从高速存储器进行故障恢复的数据开销要小很多;另一方面,FRTR 机制只需要恢复故障事务中的活跃变量,所以相比于系统 C&R 机制极大地减少了需要恢复的数据量.所以,FRTR 方法的系统状态回退开销相比于 C&R 机制要小很多,也就使 FRTR 方法的整体故障恢复开销远小于 C&R 机制.

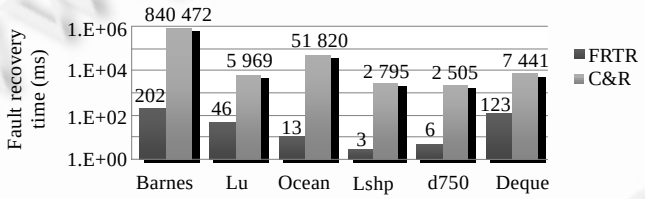


Fig.6 Comparison of fault recovery time

图 6 故障恢复时间比较

从上述实验结果可以看出:在无故障情况下,基于 FRTR 的容错事务存储系统中程序的运行时间与原事务程序的运行时间几乎相同;在有故障情况下,事务冲突不很严重的系统中,FRTR 方法大量节省了有用工作的浪费,其故障恢复开销较于经典的 C&R 技术有很大的性能优势.而即使在事务冲突严重的系统中,也会有相对较好的性能优势.

4 相关工作

自 Herlihy 在 ISCA93 上提出事务存储的概念[6],近年来,很多研究者针对事务存储的概念做了大量广泛的研究工作,提出了各种软件事务存储系统、硬件事务存储系统和混合事务存储系统的实现方案和形式化的指导设计方案.各种事务存储系统采用了特定的冲突检测机制来控制事务的并发访问,而使用特定的版本管理机制来保证事务执行的原子性.其中,典型的事务存储系统包括采用 lazy 的冲突检测机制和 lazy 的版本管理机制的全事务硬件事务存储系统 TCC[12],LogTM[10]和 UTM[11]均采用了 eager 的冲突检测机制和 eager 的版本管理机制,而 LTM[11]和 VTM[13]采用的都是 eager 的冲突检测机制和 lazy 的版本管理机制.

此外,文献[14]提出了一种基于冲突-可串行化(conflict-serializability,简称 CS)机制和多版本机制的软件事务存储系统 TSTM.相比以往基于 2 段锁实现的软件事务存储系统,TSTM 可以潜在地获得更好的性能.

而针对事务存储系统设计的正确性,文献[15]通过对事务存储的语义进行形式化的说明,不但方便了事务存储系统正确性证明,而且对事务存储系统的不同实现中事务的并发程度提供了形式化的比较.而文献[16]作为文献[15]的补充,提出了不透明性(opacity)作为评判事务存储系统实现的一个标准.文献[17]提出了可线性化(linearizability)的概念,作为评价并发对象正确性的一个条件,对于评判事务存储系统的正确性也起了重要的指导作用.

然而,这些工作都没有关注事务存储容错属性.本文挖掘了事务存储系统的容错属性,提出了基于 FRTR 的故障恢复方法容错事务存储系统,并证明了容错事务存储系统对于故障恢复的充分性.

本文对于 FRTR 方法正确性的证明,是通过证明容错事务存储系统的隔离性.在传统并发执行流的一致性问题上,已有很多广泛的研究.文献[18]就给出了传统的顺序一致性共享存储系统的执行正确性模型,提出了共享存储系统判断一个执行正确性的充要条件,即共享存储系统中的程序指令执行序与冲突访问对集的并集是无圈的.而本文对于容错事务存储系统隔离性的证明也是通过证明无环图实现的,在一定程度上借鉴了传统的并发执行流一致性问题的研究方法,但是所针对的是容错事务存储系统的事务相关关系图,为了实现容错事务存储系统的容错性质的证明.

另一方面,在传统的容错领域,rollback-recovery 技术已有广泛的研究[5],文献[5]对这些技术进行了详细的介绍.检查点技术作为一种主要的 rollback-recovery 容错技术,通常可分为系统级检查点技术和应用级检查点技术.系统级检查点技术要求周期性地将所有任务的地址空间内容(堆、栈和全局变量)、寄存器信息和通信库状态存储到稳定的存储器上,出现故障时系统回退到最近的检查点重新计算.当系统中包含大量结点时,检查点保存的开销十分巨大;而且即使在无故障时,这种保存开销也是不可避免.为减少将检查点写入稳定存储器上的开销,Plank 提出了 Diskless checkpointing 技术[19],用高速的内存代替低速的磁盘.但是,内存容量限制了这种方法在大规模科学计算中的应用.

应用级检查点是减少检查点保存数据量解决 I/O 瓶颈的一种有效方法[20],其通过由用户选择保存检查点的时机以及要保存的内容,从而减少保存和恢复检查点的 I/O 开销.为进一步减小应用级检查点的保存开销,文献[21]提出将应用级检查点保存在内存中.由于需要程序员参与检查点的保存,应用级检查点增加了程序员的编程负担.

此外,文献[22]提出的基于并行复算的故障恢复方法通过无故障处理器并行的重算失效任务实现故障恢复,从而减少了故障恢复的时间,但其根据任务间的定值-引用关系,也需要一定量的数据保存过程.

本文提出的 FRTR 故障恢复方法解决了传统容错技术不能针对性的利用事务存储系统自身的容错特性,利用事务存储系统的版本管理机制避免了额外的检查点保存开销,实现了容错事务存储系统高效的故障恢复.

5 结束语

本文提出了一种基于事务回退的事务存储系统的故障恢复机制.该方法的优点在于:

1) 充分利用了事务存储系统自身的特点,无需进行额外的检查点保存操作,从而避免了经典的 rollback-recovery 技术的检查点保存开销;
2) 根据容错事务存储系统版本管理机制的不同,对于 eager 的版本管理机制,故障恢复过程中被恢复的仅是故障事务内的活跃变量;而对于 lazy 的版本管理机制,故障恢复过程几乎可以忽略,从而进一步减少了恢复的 I/O 开销;
3) 由于故障恢复时只需要对单个事务进行回退,从而避免了无故障结点的有效工作的浪费;
4) 该方法的数据保存和故障恢复过程可完全由系统自动完成,几乎不需要用户干预.

此外,本文对容错事务存储系统的隔离性进行了讨论和证明,证明了基于事务回退的事务存储系统的故障恢复机制对于容错事务存储系统的故障恢复的充分性.

最后,本文在 simics+gems 模拟器上通过 5 个测试用例的对比实验,验证了本文的方法较于经典的检查点机制无论在无故障情况下,还是在有故障情况下,都有较明显的性能优势.

References:

[1] Harris T, Cristal A, Unsal OS, Ayguadé E, Gagliardi F, Smith B, Valero M. Transactional memory: An overview. In: Proc. of the IEEE Micro Special Issue. 2007. [doi: 10.1109/MM.2007.63]

[2] Larus J, Kozyrakis C. Transactional memory. Communications of the ACM, 2008,51(7):80−88. [doi: 10.1145/1364782.1364800]

[3] TOP500 supercomputing site. http://www.top500.org

[4] Splash-2 benchmarks site. http://www-flash.stanford.edu/apps/SPLASH/

[5] Elnozahy EN, Alvisi L, Wang YM, Johnson DB. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys, 2002,34(3):375−408. [doi: 10.1145/568522.568525]

[6] Herlihy M, Eliot J, Moss B. Transactional memory: Architectural support for lock-free data structures. In: Proc. of the 20th Annual Int'l Symp. on Computer Architecture (ISCA'93). IEEE Press, 1993. 289−300. [doi: 10.1109/ISCA.1993.698569]

[7] Volos H, Welc A, Adl-Tabatabai AR, Shpeisman T, Tian XM, Narayanaswamy R. NePalTM: Design and implementation of nested parallelism for transactional memory systems. In: Proc. of the 14th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PpoPP 2009). New York: ACM, 2009. 291−292. [doi: 10.1007/978-3-642-03013-0_7]

[8] Felber P, Fetzer C, Guerraoui R, Harris T. Transactions are back—But are they the same? "Le Retour de Martin Guerre" (Sommersby). ACM SIGACT News, 2008,39(1):47−58. [doi: 10.1145/1360443.1360456]

[9] Gray J, Reuter A. Transaction Processing: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers, 1992.

[10] Moore KE, Bobba J, Moravan MJ, Hill MD, Wood DA. LogTM: Log-based transactional memory. In: Proc. of the 20th IEEE Symp. on High-Performance Computer Architecture. 2006. [doi: 10.1109/HPCA.2006.1598134]

[11] Scott Ananian C, Asanovic K, Kuszmaul BC, Leiserson CE, Lie S. Unbounded transactional memory. In: Proc. of the 11th Int'l Symp. on High-Performance Computer Architecture (HPCA 2005). IEEE CS Press, 2005. 316−327. [doi: 10.1109/HPCA.2005.41]

[12] Hammond L, Wong V, Chen MK, Carlstrom BD, Davis JD, Hertzberg B, Prabhu MK, Wijaya H, Kozyrakis C, Olukotun K. Transactional memory coherence and consistency. In: Proc. of the 31st Annual Int'l Symp. on Computer Architecture (ISCA 2004). IEEE CS Press, 2004. 102−113. [doi: 10.1145/1028176.1006711]

[13] Rajwar R, Herlihy M, Lai K. Virtualizing transactional memory. In: Proc. of the 32nd Annual Int'l Symp. on Computer Architecture. 2005. [doi: 10.1109/ISCA.2005.54]

[14] Aydonat U, Abdelrahman TS. Serializability of transactions in software transactional memory. In: Proc. of the 2nd ACM SIGPLAN Workshop on Transactional Computing. 2008.

[15] Scott ML. Sequential specification of transactional memory semantics. In: Proc. of the 1st ACM SIGPLAN Workshop on Languages, Compilers, and Hardware Support for Transactional Computing (TRANSACT 2006). 2006.

[16] Guerraoui R, Kapalka M. On the correctness of transactional memory. In: Proc. of the 13th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. 2008. [doi: 10.1145/1345206.1345233]

[17] Herlihy MP, Wing JM. Linearizability: A correctness condition for concurrent objects. ACM Trans. on Programming Languages and Systems (TOPLAS), 1990,12(3):463−492. [doi: 10.1145/78969.78972]

[18] Hu WW,Xia PS. Out-of-Order execution in sequentially consistent shared memory systems: simulation results. Chinese Journal of Computers,1997,20(6):481−490 (in Chinese with English abstract).

[19] Plank JS, Li K, Puening MA. Diskless checkpointing. IEEE Trans. on Parallel Distrib. System, 1998,9(10):972−986. [doi: 10.1109/71.730527]

[20] Beguelin A, Seligman E, Stephan P. Application level fault tolerance in heterogeneous networks of workstations. Journal of Parallel and Distributed Computing, 1997,43(2):147−155. [doi: 10.1006/jpdc.1997.1338]

[21] Chen ZZ, Fagg GE, Gabriel E, Langou J, Angskun T, Bosilca G, Dongarra J. Fault tolerant high performance computing by a coding approach. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPoPP 2005). 2005. 213−223. [doi: 10.1145/1065944.1065973]

[22] Yang XJ, Du YF, Wang PF, Fu HY, Jia J, Wang ZY, Suo G. The fault tolerant parallel algorithm: The parallel recomputing based failure recovery. In: Proc. of the 16th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2007). 2007. [doi: 10.1109/PACT.2007.4336212]

附中文参考文献:

[18] 胡伟武,夏培肃.顺序一致共享存储系统中的乱序执行技术——基本理论.计算机学报,1997,20(6):481−490.

宋伟(1981−),男,辽宁大连人,博士生,主要研究领域为并行体系结构,容错技术.

杨学军(1963−),男,博士,教授,博士生导师,CCF 会员,主要研究领域为并行体系结构,低功耗编译,流计算,容错计算.