ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.10, October 2007, pp.2516−2527
DOI: 10.1360/jos182516

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

一种实时动态数据库故障恢复策略*

肖迎元[1+], 刘云生[2], 廖国琼[3], 邓华锋[2]

[1](天津理工大学 计算机科学与工程系,天津 300191)

[2](华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

[3](西门子有限公司 西门子中国研究院,北京 100102)

A Real-Time Dynamic Database Crash Recovery Strategy

XIAO Ying-Yuan[1+], LIU Yun-Sheng[2], LIAO Guo-Qiong[3], DENG Hua-Feng[2]

[1](Department of Computer Science and Engineering, Tianjin University of Technology, Tianjin 300191, China)

[2](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

[3](China Corporate Technology, Siemens Limited, Beijing 100102, China)

+ Corresponding author: Phn: +86-22-23679366, E-mail: xiaoyingyuan_hust@126.com

Xiao YY, Liu YS, Liao GQ, Deng HF. A real-time dynamic database crash recovery strategy. *Journal of Software*, 2007,18(10):2516−2527. http://www.jos.org.cn/1000-9825/18/2516.htm

Abstract: Distributed real-time main memory databases are usually applied in some time-critical applications. For these applications, rapid and efficient recovery in the event of site crash is very important. Firstly, through analyzing the recovery requirements of failure in distributed real-time main memory databases, the recovery correctness criteria of distributed real-time main memory database are presented. Then, a real-time dynamic crash recovery scheme based on log (RTDCRS) is presented, and the correctness of RTDCRS is proved. RTDCRS adopts the real-time logging scheme integrating the characteristics of partitioned logging, ephemeral logging and uses nonvolatile high-speed store as logging storage area in order to reduce the logging cost during the normal running. After a site crash, a dynamic recovery strategy based on classification recovery idea is adopted to decrease the downtime. Performance tests and evaluations results show that RTDCRS has better performances than the traditional recovery schemes in two aspects: The missing deadlines ratio of transactions and the time of system denying services after crashes.

Key words: distributed real-time main memory database; failure recovery; orthogonal partition; transaction class; dynamic recovery

摘 要: 分布式实时内存数据库通常使用在时间关键型应用中,对这些应用而言,故障后能迅速而有效地恢复是至关重要的.首先通过分析分布式实时内存数据库故障恢复需求,给出了其恢复正确性准则.然后提出了一种基于日志的实时动态故障恢复模式RTDCRS(real-time dynamic crash recovery scheme),并证明了其正确性.RTDCRS采用了集成分区日志和短暂日志特性的实时日志模式,同时使用非易失性高速存储设备作为日志存储区,以尽可能地降低

* Supported by the National Natural Science Foundation of China under Grant No.6073045 (国家自然科学基金); the Defense Pre-Research Project of the 'Tenth Five-Year-Plan' of China under Grant No.413150403 (国家"十五"国防预研基金)

Received 2006-06-15; Accepted 2006-08-22

系统正常运行时日志代价.在站点故障后的恢复策略上,给出了基于分类恢复思想的动态恢复策略,以尽可能地减少故障站点停止服务的时间.性能测试结果显示,RTDCRS 在事务错过截止期比率和站点停止服务时间两方面与传统的故障恢复模式相比具有明显的优越性.

关键词: 分布式实时内存数据库;故障恢复;正交分划;事务类;动态恢复

中图法分类号: TP311 **文献标识码:** A

一类应用,如工业过程控制、电网调度、军事作战指挥系统等,需要分布式实时数据库系统的支持.分布式实时数据库系统(distributed real-time database systems,简称DRTDBS)是分布式数据库系统和实时数据库系统相结合的产物,是事务和数据都可以具有定时限制的分布式数据库系统.在DRTDBS中,事务的定时限制典型地表现为事务有截止期限制.一个实时事务若在规定的截止期后(超截止期)完成,结果将变得毫无价值(对固实时事务),甚至还可能带来灾难性的后果(对硬实时事务).DRTDBS通常处理两类数据:实时数据和持久数据.实时数据仅在其有效期内有效,过期的实时数据将变为无效.持久数据在它的整个生命周期内始终有效.为了更好地满足事务和数据的定时限制,DRTDBS通常需要内存数据库技术提供支持.内存数据库(main memory databases,简称MMDB)要求数据库工作版本(main databases,简称MDB)常驻内存,外存版本(secondary databases,简称SDB)作为内存版本的备份.MMDB确保事务执行过程中的所有读、写操作都是针对内存版本,即事务执行过程中无数据I/O[1].我们将集成了MMDB技术的DRTDBS称为分布式实时内存数据库系统(distributed real-time main memory databases,简称DRTMMDBS).DRTMMDBS集成了分布式数据库、实时数据库和内存数据库的能力,但并非三者在概念、技术、机制上的简单组合,而有一系列问题需要研究和解决,如事务调度、并发控制、故障恢复等.

分布式环境本身的复杂性加上内存的易失性和脆弱性,使得相对于基于磁盘的集中式数据库而言,DRTMMDBS 发生故障的可能性更大.当系统故障发生,在恢复正常服务之前,许多实时事务可能错过它们的截止期,大量实时数据将变得无效.因此故障发生后,能迅速而有效地恢复对 DRTMMDBS 而言有至关重要的意义.

目前,对于DRTMMDBS故障恢复技术的相关研究不多,已有的工作主要是针对单一的分布式数据库、实时数据库、内存数据库的恢复策略和技术的研究.在实时数据库方面,Choi等人给出了一种采用双CPU的并行处理结构,一个CPU(database processor,简称DP)负责正常的事务处理,而另一个CPU(recovery processor,简称RP)专门负责有关恢复处理,如记日志、做检验点以及故障后进行数据库恢复.该方法提高了系统性能,但RP的利用率不高.此外,如何进行负载平衡是该结构应解决的问题[2].针对传统的基于磁盘的顺序、持久日志恢复方法的低效问题,分区日志思想[3]和短暂日志[4,5]技术分别被提了出来.分区日志采用将事务日志基于事务类(或数据类)分区存放,即不同类的事务(或数据)的日志记录在不同的分区.分区日志能够避免单一日志存储区因为严重的访问竞争而导致的系统性能瓶颈问题.然后,如何对事务(或数据)进行分类、如何确定日志分区的准则、如何确保故障后能将数据库恢复到正确、一致的状态是该策略必须加以研究和解决的问题.短暂日志不需要持久地保留日志记录,短暂日志要求事务提交时将其更新强制刷新到稳定的存储设备中,这样,事务一旦提交即可将其日志记录删除.其优点是大大缩短了故障后恢复时日志处理的时间,而缺点是缺乏持久日志记录,不利于审计跟踪.Liu[6]和Panda[7]分别研究了故障后加快恢复处理速度的技术,但这些技术都需要在整个故障恢复期间停止系统服务.针对内存数据库,基于影子的恢复技术被提了出来[8−10],该技术的优点是消除了日志开销,恢复速度快,缺点是在事务生命周期内数据库需维持其更新数据页的两个版本——当前页和影子页,同时需维护大量的页表指针.

本文通过分析 DRTMMDBS 的故障恢复需求,给出了 DRTMMDBS 恢复正确性准则.在此基础上,提出了适合 DRTMMDBS 的充分考虑了数据和事务的定时限制的基于日志的动态恢复模式.

1 故障恢复需求和恢复正确性准则

在分析 DRTMMDBS 的故障恢复需求之前,我们先给出如下定义:

定义 1. 实时数据对象 X 定义为一个三元组:$X::=\langle V(X),ST(X),VI(X)\rangle$.其中,$V(X)$表示 X 的当前状态或值;$ST(X)$表示采样时标,即采样 X 对应的外部世界对象的时间;$VI(X)$表示 X 的有效期,即自 $ST(X)$算起 $V(X)$具有有效性的时间长度.

根据上述定义,持久数据可看成是 $VI(X)$取无穷大时的实时数据.

定义 2. 一个数据对象X被称为满足时态一致性,如果有:$ST(X)+VI(X)\geq T_c$.这里,T_c表示当前时刻.

DRTMMDBS中事务和数据的定时限制及分布特性对恢复策略、技术提出了下述需求[11]:

(1) 恢复必须考虑数据的定时限制和存取频率,如有效期紧迫的数据及存取频率高的"热点"数据应优先恢复;

(2) 恢复也需要考虑事务的定时限制,如高优先级事务所需存取的数据应优先恢复;

(3) 恢复除了要考虑数据库的逻辑(内部)一致性以外,数据的时态一致性也必须被确保;

(4) 除了数据库状态的恢复以外,由夭折事务导致的外部世界状态的改变也必须被恢复;

(5) 恢复必须结合一定的提交协议以确保分布式实时事务故障的原子性;

(6) 恢复必须考虑内存数据库的特性和要求;

(7) 快速且有效的恢复,以保证故障发生后事务错过截止期的比率并不会显著增加.

基于上述故障恢复需求,我们给出了 DRTMMDBS 的如下恢复正确性准则:

准则 1(实时数据恢复准则). 在故障恢复时,对于实时数据对象X,若$ST(X)+VI(X)\leq T_c$,则对X的所有更新操作,无论对应的事务提交与否,都无须进行REDO或UNDO恢复,而通过重新从外界环境采样进行恢复.

准则 1 要求对于过期的实时数据采用重新从外界环境采样进行恢复,以确保数据的时态一致性.

DRTMMDBS 通过两种方式直接与外部世界交互作用:一是关于外部世界状态或事件的信息被记录到数据库中;二是事务可以启动各种影响外部世界的活动.为此,在 DRTMMDBS 中有如下事务分类:

(1) 数据采样事务:记录外部世界的状态或发生的事件到数据库中.它是简单的只写事务.

(2) 数据处理事务:类似于传统数据库中的事务,用来执行用户或应用程序发起数据库操作请求.

(3) 控制事务:能触发外部世界中相关活动的事务.控制事务通过向受控子系统发送控制消息,来触发改变外部世界的活动.

在 DRTMMDBS 中,控制事务的执行将引发外部世界相应的活动.对于已提交的控制事务引发的外部世界状态的改变无须恢复,而对夭折的控制事务所引发的改变外部世界状态的活动,则必须执行相应的补偿或替代事务进行复原.

下文中用符号 AT 表示控制事务 T 所引发的活动;AT'表示 AT 的补偿或替代事务.

准则 2(外部世界状态恢复准则). 若控制事务 T 夭折,AT 已发生,则执行 AT'以恢复外部世界状态.

分布式实时事务的执行被分解为多个分布在不同站点的实时子事务的协同执行,因此要确保分布式实时事务的原子性就必须保证构成分布式实时事务的所有实时子事务要么全提交,要么全不提交,即使故障发生.

假定分布式实时事务$T=\{ST_1,ST_2,\ldots,ST_i,\ldots,ST_m\}$,这里,$ST_i(1\leq i\leq m)$表示构成$T$的在站点$i$执行的实时子事务.

准则 3(分布式实时事务REDO恢复准则). 在故障时刻,若分布式实时事务T已提交,则在故障站点i根据需要执行REDO恢复操作以确保ST_i的效果.

对于准则 3,下述两种情况,无须执行 REDO 恢复:

(1) ST_i所更新的数据对象的后映像已被物理地写入数据库;

(2) ST_i所更新的数据对象满足$ST(X)+VI(X)\leq T_c$.

对于(2),通过执行采样事务进行恢复.

准则 4(分布式实时事务UNDO恢复准则). 在故障时刻,若分布式实时事务T未提交,则对所有的ST_i $(1\leq i\leq m)$在相应站点根据需要执行UNDO恢复.

对于准则 4,下述两种情况,ST_i无须执行UNDO恢复:

(1) ST_i所更新的数据对象的后映像还未被物理地写入数据库;

(2) ST_i所更新的数据对象满足$ST(X)+VI(X)\le T_c$.

2 实时动态故障恢复模式

就故障恢复而言,DRTMMDBS 需要处理各种不同类型的故障,如通信故障、网络分割、站点故障(系统故障)等.本文主要考虑站点故障(crash).站点故障造成故障站点易失性主存数据丢失(MDB 遭到破坏),系统非正常终止,从而导致一些已提交事务(效果已写入 MDB,但未写出到 SDB)的效果丢失.同时,站点故障可能影响到分布式实时事务的原子性.此外,在恢复服务之前,站点故障还可能导致许多实时事务错过它们的截止期,大量实时数据变得无效.

在前面分析的 DRTMMDBS 故障恢复需求和恢复正确性准则的基础上,针对站点故障,本文提出了一种实时动态故障恢复模式(real-time dynamic crash recovery scheme,简称 RTDCRS).

2.1 实时日志模式

传统的顺序日志模式将日志记录存储在单一的日志文件中,日志文件会因为严重的访问竞争而成为系统性能的瓶颈,这显然不能满足 DRTMMDBS 对实时性能的需求.本文给出了基于非易失 RAM(random access memory)的、集成分区日志和短暂日志特性的实时日志模式,后文中将该日志模式简记为 NRPERTL.

2.1.1 基于分区日志的事务类的划分

NRPERTL 采用了日志存储区分区的思想,即根据事务类的划分策略,将日志存储区分成对应的不同分区,属于不同类别的事务的日志记录被分别存储在不同的日志分区,从而可以避免单一日志文件导致的系统性能瓶颈.

分区日志根据事务类的划分策略将日志存储区分成对应不同分区,因此,事务分类策略就决定了日志的分区方法.在 DRTMMDBS 中,事务具有丰富的语义特征,可以按不同的特征来进行分类,本文从日志分区的角度来考虑事务划分策略.单一的分区日志技术对事务类的划分策略并无太多的要求,然而 RTDCRS 在故障后的恢复处理策略上采用了分类恢复的思想,为了确保故障后能将数据库恢复到正确、一致性状态,基于分区日志的事务类划分就必须遵循一定的原则.

下面的定义中,用ST表示所有可能事务的集合;ST_i表示ST的一个子集;$DS(T)$表示事务T存取的数据对象的集合;$DS(ST_i)$表示ST_i中所有事务可能存取的数据对象的集合;LS表示日志存储区;$SA(T)$表示事务T的日志记录的存储区域.

定义 3. 假定$ST_1,ST_2\subseteq ST$,若$ST_1\cap ST_2=\varnothing$,则称$ST_1$和$ST_2$互斥.记为$ST_1\perp ST_2$.

定义 4. 假定$ST_1,ST_2,\ldots,ST_n\subseteq ST$,若对任意的$ST_i,ST_j,1\le i\le n,1\le j\le n$,有$ST_i\perp ST_j$,则称集合$\{ST_1,ST_2,\ldots,ST_n\}$为$ST$的互斥集.

定义 5. 假定$\{ST_1,ST_2,\ldots,ST_n\}$为一互斥集,若有$ST_1\cup ST_2\cup\ldots\cup ST_n=ST$,则称$\{ST_1,ST_2,\ldots,ST_n\}$为$ST$的一个分划(分类).其中,$ST_i,1\le i\le n$,称为一个事务类.

定义 6. 假定$LS_1,LS_2,\ldots,LS_n\subseteq LS$,若对任意的$LS_i,LS_j,1\le i\le n$,有$LS_i\cap LS_j=\varnothing$,则称集合$\{LS_1,LS_2,\ldots,LS_n\}$为$LS$的互斥集.

定义 7. 假定$\pi=\{ST_1,ST_2,\ldots,ST_n\}$为$ST$的一个分划,$\mu=\{LS_1,LS_2,\ldots,LS_n\}$为$LS$的一个互斥集,若存在双射(一一映射)$f:\pi\to\mu$,则称$\mu$为关联分划$\pi$的$LS$的一个分区,$LS_i(1\le i\le n)$称为一个子日志区.其中,$f$表示$\forall ST_i\in\pi$有且只有一个$LS_i\in\mu$使得$\forall T\in ST_i$都有$SA(T)\subseteq LS_i$,且反之也成立.

定义 8. $\{ST_1,ST_2,\ldots,ST_n\}$被称为$ST$的一个正交分划,若$\{ST_1,ST_2,\ldots,ST_n\}$为$ST$的一个分划且$\forall ST_i,ST_j\in\{ST_1,ST_2,\ldots,ST_n\}$,有$DS(ST_i)\cap DS(ST_j)=\varnothing$.

原则 1. 为了确保故障后能将数据库恢复到正确、一致性状态,RTDCRS 要求日志分区所关联的 ST 的分划$\pi=\{ST_1,ST_2,\ldots,ST_n\}$满足条件:$\exists\pi_1=\{ST_{k_1},ST_{k_2},\ldots,ST_{k_j}\}\subset\pi,(DS(\pi_1)\cap DS(\pi_2))=\varnothing$.这里,$\pi_2=ST-\pi_1$.

我们称满足原则 1 中条件的分划为规范化分划.

定理 1. 若$\{\pi_1,\pi_2\}$是ST的一个正交分划,$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}\}$是$\pi_1$的一个分划,$\{ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}$是$\pi_2$的一个分划,则$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}, ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}$为一规范化分划.

证明:因为$\{\pi_1,\pi_2\}$是一正交分划,所以有$DS(\pi_1)\cap DS(\pi_2)=\varnothing$.因此,要证明$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}, ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}$为一规范化分划,根据规范化分划的定义,只需证明$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}, ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}$为一分划.由$\{\pi_1,\pi_2\}$是一正交分划可得$\pi_1\cup\pi_2=ST,\pi_1\cap\pi_2=\varnothing$.又因为$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}\}$是$\pi_1$的一个分划,所以$ST_{k_1}\cup ST_{k_2}\cup\ldots\cup ST_{k_j}=\pi_1$且$\forall ST_u,ST_v\in\pi_1,ST_u\cap ST_v=\varnothing$,同理,有$ST_{m_1}\cup ST_{m_2}\cup\ldots\cup ST_{m_i}=\pi_1$且$\forall ST_u,ST_v\in\pi_2,ST_u\cap ST_v=\varnothing$.由此可得$ST_{k_1}\cup ST_{k_2}\cup\ldots\cup ST_{k_j}\cup ST_{m_1}\cup ST_{m_2}\cup\ldots\cup ST_{m_i}=ST$且$\forall ST_u,ST_v\in\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}, ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}, ST_u\cap ST_v=\varnothing$,即$\{ST_{k_1}, ST_{k_2},\ldots, ST_{k_j}, ST_{m_1}, ST_{m_2},\ldots, ST_{m_i}\}$为$ST$的一个分划.从而定理得证. □

根据定理 1,可以容易地构造出规范化分划.下面具体给出一种构造方法.首先将ST按事务优先级划分成两个子集:$ST_{>p}=\{T_i|\Pr(T_i)>p,T_i\in ST\}$和$ST_{<=p}=\{T_i|\Pr(T_i)\le p,T_i\in ST\}$,这里,$\Pr(T_i)$表示事务$T_i$的优先级.显然,$\{ST_{>p},ST_{<=p}\}$为$ST$的一个分划.通过适当地调整$p$的值,可使$DS(ST_{>p})\cap DS(ST_{<=p})=\varnothing$(最坏的情况$ST_{>p}$和$ST_{<=p}$中有一个为$\varnothing$),即$\{ST_{>p},ST_{<=p}\}$为$ST$的一个正交分划.$DS(ST_{>p})$为属于$ST_{>p}$中的高优先级事务所存取的数据对象的集合,我们称$DS(ST_{>p})$为关键数据类,称$ST_{>p}$为关键事务类;$DS(ST_{<=p})$为属于$ST_{<=p}$中的低优先级事务所存取的数据对象的集合,我们称其为一般数据类,称$ST_{<=p}$为一般事务类.进一步地,我们将$ST_{>p}$划分成两个(也可为多个)子集:$ST_{>p,>f}=\{T_i|Fr(T_i)>f,T_i\in ST_{>p}\}$和$ST_{>p,<=f}=\{T_i|Fr(T_i)\le f,T_i\in ST_{>p}\}$,这里,$Fr(T_i)$代表$T_i$的估计执行频率,$f$为一设定的频率值.显然,$\{ST_{>p,>f},ST_{>p,<=f}\}$是$ST_{>p}$的一个分划.类似地,可将$ST_{<=p}$划分成两个(也可为多个)子集:$ST_{<=p,>f}=\{T_i|Fr(T_i)>f,\ T_i\in ST_{<=p}\}$和$ST_{<=p,<=f}=\{T_i|Fr(T_i)\le f,T_i\in ST_{<=p}\}$.显然,$\{ST_{<=p,>f},ST_{<=p,<=f}\}$是$ST_{<=p}$的一个分划.令$\sigma=\{ST_{>p,>f},ST_{>p,<=f},\ ST_{<=p,>f},ST_{<=p,<=f}\}$,则根据定理 1,$\sigma$是一个规范化分划.后文中,NRPERTL和动态恢复策略的描述都是基于σ.

2.1.2 存储介质的组织

如图 1 所示,每个站点的存储介质分为 3 层:磁盘存储器、非易失RAM、易失RAM(主存).本地外存数据库(local secondary databases,简称LSDB)被存储在磁盘存储器中.非易失RAM作为日志存储区,对应于规范化分划σ,整个日志存储区被划分成 4 个独立的子日志区:关键高频事务类子日志区、关键低频事务类子日志区、一般高频事务类子日志区和一般低频事务类子日志区,分别简记为:LS_{ch},LS_{cl},LS_{gh}和LS_{gl},它们分别用来对应地存储$ST_{>p,>f}$,$ST_{>p,<=f}$,$ST_{<=p,>f}$和$ST_{<=p,<=f}$事务类的日志记录,即$\{LS_{ch},LS_{cl},LS_{gh},LS_{gl}\}$是关联规范化分划$\sigma$的日志存储区的一个分区.我们将$LS_{ch}$和$LS_{cl}$统称为关键日志区;$LS_{gh}$和$LS_{gl}$统称为一般日志区.本地内存数据库(local main databases,简称LMDB)被存储在易失性主存中,并且被分成两个独立的分区:关键数据分区和一般数据分区,用来分别存储关键数据类和一般数据类.为了充分利用非易失性日志存储空间,我们采用空间动态分配策略,即在系统初始化时为每一子日志区分配一定数量的日志存储区空间,在系统运行过程中,当某一子日志区没有足够的空间存储对应事务类的日志记录时,系统为它分配一个新的日志页,属于同一子分区的日志页被链接在一起.

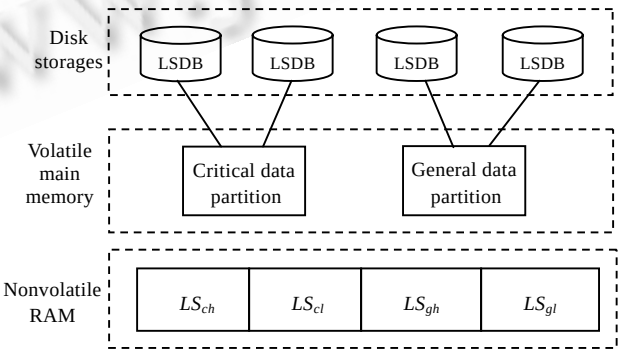


Fig.1 Storage media organization at a site

图 1 单个站点存储介质的组织

2.1.3 实时日志记录类型与结构

数据库更新策略(包括延迟更新和立即更新)对日志处理有很大的影响.立即更新可能产生“脏数据”,从而导致串联夭折(cascading abort),严重影响系统性能.因此,DRTMMDBS 通常采用延迟更新策略.对于延迟更新策略,由一事务更新了的数据对象的值直到该事务提交才真正反映到 LMDB.因此,若采用延迟更新策略,故障后恢复处理时,UNDO 操作不再需要.后文中,我们假定延迟更新策略被采用.

结合实时提交协议 1PRCP(one-phase real-time commit protocol)[12],我们设计 5 种日志记录类型:Begin,Redo,Compensate,Ready和Commit,如图 2 所示.其中,*P-TID*表示分布式事务(全局事务)标识,定义为*P-TID*::=⟨*Cor_Addr*,*N*⟩,这里,*Cor_Addr*表示分布式事务的协调者的网络地址;*N*代表分布式事务的序号,在每个协调者站点,*N*是单调递增的.对于本地事务,*P-TID*被置为*NULL*.*TID*为本地事务或子事务标识,定义为*TID*::=⟨*Addr*,*SN*⟩,这里,*Addr*表示本地事务或子事务所在站点的网络地址;*SN*表示本地事务或子事务的序号,在每个站点*SN*是单调递增.*B*,*D*,*CP*,*R*和*C*用来分别表示Begin,Redo,Compensate,Ready和Commit这 5 种日志记录类型.*TS*表示Redo或Compensate或Commit类型日志记录创建时刻的逻辑时标.在每个站点,逻辑时标的初始值置为 0,每当一条Redo或Compensate或Commit日志记录或后文介绍的检验点日志记录被创建,逻辑时标的值增加1.*RID*代表被更新数据对象的标识.*BN*表示与LMDB中数据页相对应的LSDB中数据块的逻辑块号.*AI*表示被更新数据对象的后映像.*VTI*表示实时数据对象的最大有效时刻,即*VTI*=*ST*(*X*)+*VI*(*X*).对持久数据对象而言,其*VTI*置为无穷大.*CA*表示控制事务所对应的补偿活动.

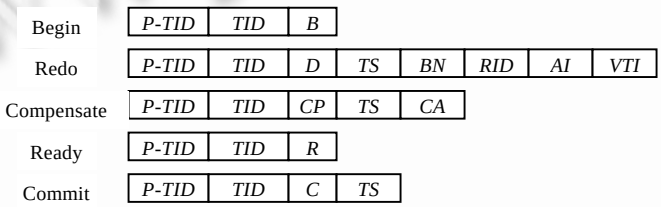


Fig.2 Types and structures of real-time log record

图 2 实时日志记录的类型与结构

当一个事务开始执行时,在执行站点,一条 Begin 日志记录在对应的子日志区被创建;对每一个更新操作,对应的 Redo 日志记录被创建;当一个控制事务向受控子系统发送控制信息时,相应的 Compensate 日志记录被创建;在参与者站点,当一个子事务进入了准备提交状态且未超截止期,一条相应的 Ready 日志记录在该参与者站点被创建;当一个分布式事务的协调者在限定时间内收到来自所有参与者的 Vote-Commit 消息时,在协调者站点,相应的 Commit 日志记录被创建;当一个子事务收到来自协调者的 Global Commit 决定时,在其所在站点,相应的 Commit 日志记录被创建.

2.2 本地检验点模式

检验点的目的是在辅助存储设备中维持数据库的最近版本、确定恢复起始点以及限制故障恢复时间.在RTDCRS中,每个站点独立地执行本地检验点(在后文中,检验点指的都是本地检验点).在检验点执行过程中,LMDB的更新被写出到LSDB,同时,一些无用的日志记录被删除以释放相应的日志存储空间.由于检验点的执行涉及到大量的磁盘I/O,故检验点模式(静态或模糊)和执行频率对系统性能有很大的影响.静态检验点要求在将LMDB的更新写出到LSDB的过程中不允许存在活跃的事务,这显然不利于DRTMMDBS中事务和数据的时限满足,因此,RTDCRS采用模糊检验点模式.此外,在检验点触发时机上,RTDCRS不是采用固定时间间隔的周期性触发策略,而是根据日志存储区空间使用率(简记为R_{LU})是否到达设定的阈值α来决定是否触发检验点.当R_{LU}大于α时,检验点被触发执行.α的初始值可根据应用的要求预先设定,在系统运行过程中可以通过修改α的值来动态调整检验点执行的频率.

当检验点执行时,检验点日志记录被写入系统在非易失性 RAM 中为检验点日志记录分配的独立存储空

间.检验点日志记录包括 5 个部分:检验点位域(*CKB*)、关键事务类恢复起始时标(*CTRST*)、一般事务类恢复起始时标(*GTRST*)、检验点时标(*CKT*)和更新页域(*UPF*).*CKB* 用来表示本次检验点是否成功完成:*CKB* 取值为 1 表示成功完成,为 0 表示检验点执行过程中发生系统故障.*CTRST* 表示故障后需要执行 REDO 恢复操作的关键事务类的最小 *TS*(逻辑时标值),而 *GTRST* 表示故障后需要执行 REDO 恢复操作的一般事务类的最小 *TS*(逻辑时标值).*CKT* 表示检验点被触发时的逻辑时标.*UPF* 为 LMDB 中的每一数据页设置一个更新位(updating bit)来记录该页的更新状态,该位为 0 表示自上次检验点后该页还未被更新;该位为 1 表示自上次检验点后该页已被更新.检验点执行过程可描述如下:

Procedure *LocalCheckpoint*()

1: *CKB*=0;

2: *CKT=TCounter++*;

3: Write data pages updated by committed transaction out to LSDB;

4: Set *UPF* according to updating state of data page;

5: Write the data pages whose upadting bits are 1 out to LSDB;

6: Delete useless log records, i.e. the log records of aborted transactions or whose logic timestamp in Commit log records is not larger than *CKT*, and free the corresponding logging store spaces;

7: Get the minimal logging timestamp of critical transaction class, set the value of *CTRST*;

8: Get the minimal logging timestamp of ordinary transaction class, set the value of *GTRST*;

9: *CKB*=1;

在上面的描述及后面的恢复处理的算法描述中,*TCounter* 用来表示逻辑时标计数器,其初始值置为 0,每当一条 Redo 或 Compensate 或 Commit 或检验点日志记录被创建,其值加 1.系统在非易失性 RAM 中为 *TCounter* 分配相应的存储空间.

2.3 动态恢复处理策略

在站点故障后的恢复处理过程中,首先,LSDB 被装入主存并重建 LMDB;然后,恢复子系统负责将 LMDB 恢复到最近的一致性状态.为了提高系统性能,RTDCRS 给出了基于分类恢复思想的动态恢复处理策略,动态恢复处理策略以前面提出的 NRPERTL 日志模式为基础,其关键特征是:关键数据类被首先恢复,在恢复一般数据类之前恢复站点服务,然后边服务边恢复一般数据类,以最大限度地降低故障站点停止服务的时间.结合实时提交协议 1PRCP,动态恢复策略能够确保分布式事务故障的原子性.动态恢复策略具体包括如下几个步骤:

(1) 从 LSDB 中将关键数据类装入主存相应数据分区,重建 LMDB;

(2) 消除属于关键事务类的夭折的控制事务导致的对外部世界改变,并将关键数据类恢复到最近一致性状态;

(3) 恢复故障站点的服务;

(4) 从 LSDB 中将一般数据类装入主存相应数据分区;

(5) 消除属于一般事务类的夭折的控制事务导致的对外部世界改变,并将一般数据类恢复到最近一致性状态.

在上面的步骤中,步骤(2)和步骤(5)的实现算法可统一描述如下:

Procedure *CrashRecovery*(*char dc*)

Input: *dc* denotes the data class asking to be recovered, and *dc*="*C*" stands for critical data class, while *dc*="*G*" stands for general data class.

1: *FT=TCounter*;

2: **if** (*dc*="*C*") **then**

3: *RST=CTRST*;

4: Scan the critical logging partition (including LS_{ch} and LS_{cl}) to look for all Compensate log reords

$CPLog_{T_i}$, which satisfy the following conditions: The corresponding Commit log records $CommitLog_{T_i}$ don't exist, i.e. the corresponding transactions don't commit successfully, and timestamp of $CPLog_{T_i}$ is smaller than *FT*. At the same time insert these $CPLog_{T_i}$ into compensating activity recovery list (CARL) in order of their timestamp;

5: **else**

6: *RST*=*GTRST*;

7: Scan the general logging partition (including LS_{gh} and LS_{gl}) to look for all Compensate log reords $CPLog_{T_i}$, which satisfy the following conditions: the corresponding Commit log records $CommitLog_{T_i}$ don't exist, i.e. the corresponding transactions don't commit successfully, and timestamp of $CPLog_{T_i}$ is smaller than *FT*. At the same time insert these $CPLog_{T_i}$ into compensating activity recovery list (CARL) in order of their timestamp;

8: Scan reversely CARL until the head of CARL and for each $CPLog_{T_i}$, execute the corresponding compensating activity $CPLog_{T_i}.CA$;

9: **while** (*RST*≤*FT*)

10: **if** (*dc*="*C*") **then**

11: Scan the critical logging partition to look for the Redo log record $RedoLog_{T_{i,k}}$ whose timestamp is RST;

12. **else**

13: Scan the general logging partition to look for the Redo log record $RedoLog_{T_{i,k}}$ whose timestamp is *RST*;

14: **if** (find the Redo log record $RedoLog_{T_{i,k}}$) **then**

15: **if** (find the corresponding Commit log record $CommitLog_{T_i}$) **then**

16: **if** ($RedoLog_{T_{i,k}}.VTI > T_c$) **then**

17: *REDO*($RedoLog_{T_{i,k}}$)

18: *RST*++; continue;

19: **else**

20: *RST*++;

21: Triggeer the corresponding sample transaction to update overdue data;

22: **if** (find the corresponding Ready log record $ReadyLog_{T_i}$) **then**

23: **if** (receive Commit message) **then**

24: *REDO*($RedoLog_{T_{i,k}}$);

25: Add $CommitLog_{T_i}$ to the corresponding logging partition;

26: *RST*++;

27: **if** (receive Abort message) **then**

28: *RST*++;

29: **else**

30: *RST*++;

31: **else**

32: *RST*++;

33: **endwhile**

在上面的算法描述中, $RedoLog_{T_{i,k}}$ 表示事务T_i的第k条REDO日志记录;过程*REDO*($RedoLog_{T_{i,k}}$)负责将 $RedoLog_{T_{i,k}}$ 记录的数据对象在LMDB中的值用其后映像覆盖; $RedoLog_{T_{i,k}}.VTI$表示 $RedoLog_{T_{i,k}}$ 记录的数据对

象的有效时刻.由于实时数据对象一旦过期,则其值即变得无意义,因此,在上述动态恢复处理策略中,对过期的实时数据对象并不执行 REDO 操作,还是采取触发相应的采样事务来更新过期数据对象的值(遵循实时数据恢复准则).

2.4 RTDCRS的正确性

在传统的顺序(非分区)日志模式中,日志记录按对应操作的执行顺序依次记录在单一的日志文件中.基于顺序日志模式的数据库系统,故障后恢复时,首先从日志文件尾开始反向扫描日志文件,执行 UNDO 操作;然后从恢复起始点开始,正向扫描日志文件,执行REDO操作.只要系统正常运行时日志登记遵循先写日志原则,则上述恢复处理就能确保将数据库恢复到最近的一致性状态.而 RTDCRS 采用了动态恢复处理策略且日志记录基于事务类分区存放.显然,RTDCRS 已有别于采用静态恢复处理策略(整个数据库恢复完成后才开始提供服务)、基于顺序日志的传统恢复模式,因此其正确性必须得以证明.

引理 1. 假定$\{ST_1,ST_2\}$是系统中事务集合ST的一个正交分划,若系统正常运行时所有属于事务类ST_1的事务的日志按执行顺序依次记录在日志存储区Log_1,所有属于事务类ST_2的事务的日志按执行顺序依次记录在日志存储区Log_2,则系统故障后,先依据Log_1对属于事务类ST_1的事务进行恢复,再依据Log_2对属于事务类ST_2的事务进行恢复所得数据库的状态等同于依据单一日志文件Log进行恢复得到的效果.

证明:因为$\{ST_1,ST_2\}$是ST的一个正交分划,故有$DS(ST_1)\cap DS(ST_2)=\varnothing$.设$X$是故障后需进行恢复(执行UNDO或REDO恢复操作)的任一数据对象,则要么$X\in DS(ST_1)$,要么$X\in DS(ST_2)$.不失一般性,设$X\in DS(ST_1)$.假定对X需进行恢复的更新操作的序列为$\{U_1,U_2,\ldots,U_m\}$,设$\{U_1,U_2,\ldots,U_m\}$对应的更新日志记录的序列为$\{L_1,L_2,\ldots, L_m\}$.由引理 1 的条件可知,$\{L_1,L_2,\ldots,L_m\}$全部位于Log_1日志存储区,而若采用单一日志文件模式,则$\{L_1,L_2,\ldots,L_m\}$位于日志文件Log.因此,采用分类恢复模式(先依据Log_1对属于事务类ST_1的事务进行恢复,再依据Log_2对属于事务类ST_2的事务进行恢复)对X进行恢复得到的值,就是依据日志记录的序列$\{L_1,L_2,\ldots,L_m\}$得到的效果,而采用单一日志文件Log对X进行恢复得到的值同样是依据日志记录的序列$\{L_1,L_2,\ldots,L_m\}$得到的效果,结论成立. □

定理 2. RTDCRS 能够确保将数据库恢复到最近的一致性(包括内部一致性和时态一致性)状态.

证明:(1) 首先证明RTDCRS能够确保数据库的内部一致性.假定$\pi=\{ST_1,ST_2,\ldots,ST_n\}$是RTDCRS所采用的规范化分划,$\mu=\{LS_1,LS_2,\ldots,LS_n\}$为关联$\pi$的$LS$的日志分区.根据规范化分划的定义,可以找到$\pi_1=\{ST_i,ST_{i+1},\ldots,ST_{i+k}\}\subset\pi$,使得$DS(\pi_1)\cap DS(\pi-\pi_1)=\varnothing$,即$\{\pi_1,(\pi-\pi_1)\}$为$ST$的一个正交分划.令$\mu_1=\{LS_i,LS_{i+1},\ldots,LS_{i+k}\}$,$\mu_2=\{LS_1,LS_2,\ldots,LS_{i-1},LS_{i+k+1},LS_{i+k+2},\ldots,LS_n\}$,即$\mu_1$关联$\pi_1$,$\mu_2$关联$\pi-\pi_1$.当系统故障发生,RTDCRS采用基于分类恢复的动态恢复策略,不失一般性,假定故障发生后,数据类$DS(\pi_1)$中的数据首先被恢复,然后系统边服务边恢复数据类$DS(\pi-\pi_1)$中的数据.在具体恢复某一数据类时,恢复算法依据与该数据类相关联的日志区域中日志记录的逻辑时标能够确保对该类数据按正确的顺序执行恢复操作,即对于需进行恢复处理的所有补偿日志记录按照逻辑时标从大到小的顺序依次执行相应的补偿活动以消除对外部环境状态的改变;对需进行重做操作的所有更新日志记录按照逻辑时标从小到大的顺序依次执行相应的REDO操作(因采用了延迟更新策略,故无须执行UNDO恢复),即对$DS(\pi_1)$或$DS(\pi-\pi_1)$的恢复效果就等同于$DS(\pi_1)$或$DS(\pi-\pi_1)$采用单一日志区的效果.依据引理1,先恢复$DS(\pi_1)$再恢复$DS(\pi-\pi_1)$的效果就等同于采用单一日志文件的顺序日志恢复模式的效果,而采用单一日志文件的顺序日志恢复模式能够确保将数据库恢复到最近的内部一致性状态,也就是证明了RTDCRS能够确保数据库的内部一致性.

(2) 其次证明RTDCRS能够确保数据库的时态一致性.根据RTDCRS的恢复处理算法,对每一个需执行REDO恢复的数据对象X,若有$VTI(X)>T_c$,即X在恢复时刻还未过期,则执行REDO恢复;若有$VTI(X)\le T_c$,即X在恢复时刻已过期或已到期,则触发相应的采样事务更新X的值.因此,总能确保恢复后的X满足:$VTI(X)=(ST(X)+VI(X))\ge T_c$,即RTDCRS能够确保数据库的时态一致性. □

除了确保数据库的一致性以外,RTDCRS 还必须满足第 2 节提出的 DRTMMDBS 恢复正确性准则.

定理 3. RTDCRS 满足实时数据恢复正确性准则.

证明:由于RTDCRS采用了延迟更新策略,故对未提交事务的更新无须执行UNDO恢复.在基于RTDCRS的

恢复算法中,对由提交事务更新过的数据对象X,若在恢复时刻X已过期,即满足$ST(X)+VI(X))\leq T_c$,则不进行REDO恢复,还是通过执行采样事务进行恢复,因此,TCCRS满足实时数据恢复正确性准则(准则 1). □

定理 4. RTDCRS 能够确保外部世界状态的正确恢复(满足准则 2).

证明:首先,RTDCRS 在日志记录类型上增加了 Compensate 日志记录类型,Compensate 日志记录中包含了引起外部世界状态改变的活动的补偿活动.其次,在分类恢复策略的实现算法中,对所有已触发了活动但由于故障未成功提交的控制事务,依据 Compensate 日志记录的逻辑时标从大到小的顺序依次执行相应的补偿活动,从而能够正确地恢复外部世界的状态,确保了外部世界状态的正确恢复. □

定理 5. RTDCRS 满足分布式实时事务恢复正确性准则(满足准则 3、准则 4).

证明:采用了延迟更新策略,因此对未提交事务的更新无须执行UNDO恢复.在基于RTDCRS的恢复算法中,对已提交的分布式实时事务T的在故障站点上执行的子事务ST_i所更新的数据对象X,根据在恢复时刻X是否已过期来决定执行REDO恢复还是通过执行采样事务进行恢复,因此,RTDCRS满足分布式实时事务恢复正确性准则. □

3 RTDCRS 的性能测试与评估

恢复模式性能的优劣主要取决于:1) 系统正常运行时日志的代价;2) 故障发生后系统停止服务的时间(T_{DT}).我们在自行开发的分布式实时内存数据库管理系统原型ARTs-II上完成了对RTDCRS的性能测试.我们首先比较了不同日志模式运行时代价对系统性能的影响;然后测试了分区数对NRPERTL性能的影响;最后对系统停止服务的时间T_{DT}进行了评估.

性能测试的主要参数设置见表 1,其中,$U[i,j]$表示服从区间$[i,j]$上的均匀分布的随机变量.在我们的实验中,全局内存数据库(global main databases,简称 GMDB)为由 50 000 个数据页组成的集合,这些数据页被分配到 5 个站点形成相应的本地内存数据库,对数据的访问和并发控制都发生在页级,并发控制策略采用分布式高优先级两段锁协议(distributed high priority-2 phase lock,简称 DHP-2PL),事务优先级分派采用最早截止期优先策略(earliest deadline first,简称 EDF).事务 T 的截止期按如下公式计算:$Deadline(T)= AT(T)+Slack\times ET(T)$.这里,$Deadline(T)$表示 T 的截止期;$AT(T)$表示 T 到达系统的时间;$Slack$ 为松弛因子,通常为一个满足均匀分布的随机变量;$ET(T)$表示 T 的估计执行时间.$ET(T)$按如下公式估算:$ET(T)=OpN\times AET$,其中,OpN 和 AET 的含义见表 1.我们以实时事务错过截止期的比率 MDR(missing deadline ratio)为主要性能指标,其定义如下:MDR=(错过截止期事务的数目)/(进入系统事务的总数).显然,MDR 越小,系统实时性能越好.

Table 1 Experiment parameters

表 1 实验参数

Parameters	Value (unit)	Description
NS	5	Number of site of DRTMMDBS
S_{LMDB}	10 000 (page)	Size of LMDB at each site
R_{CD}	0.4	Ratio of critical data class in total data
R_{TD}	0.8	Ratio of real-time data in total data
NP	4	Number of logging storage partitions
SL	8 (MB)	Size of logging storage area at each site
AET	0.4 (ms)	Average execution time per transaction operation
PU	0.4	Probability of a transaction operation to be update operation
$Slack$	U[2.0,6.0]	Slack factor
NTO	U[4,8]	Number of operations contained in a transaction
α	0.8	Threshold of the utilization rate of logging storage area

实验中将RTDCRS的NRPERTL日志模式与另外 3 种日志模式——无日志模式(non-logging scheme,简称NLS)、基于磁盘的分区日志模式(partitioned logging scheme based on disk,简称PLSD)和基于磁盘的顺序日志模式(sequential logging scheme based on disk,简称SLSD)进行比较.NLS表示系统运行过程中不记录事务日志,显然这不能满足恢复的要求,这里将它作为不同模式代价比较的基准.图 3 显示了在分区数为 4 情况下的实验结果.显然,从图 3 可以看出,当事务的到达率提高时,4 种日志模式所对应的MDR都相应增大,而NRPERTL最接近

NLS的效果,它的*MDR*明显地小于其他两种日志模式.在同样的条件下,对于分区数取 2,6 及 8 的情况进行了实验,也得到了相似的结果.此外,我们还在改变某些参数取值的情形下(如改变α的取值或R_{TD}或R_{CD}的取值)来进一步将NRPERTL与其他日志模式进行实验比较,结果显示,我们提出的NRPERTL在确保实时事务截止期的满足上都显示了明显的优越性(受篇幅所限,本文未给出相应的实验结果图).

日志分区数是影响 NRPERTL 性能的重要因素之一,多分区能够改进并发事务对日志文件尾的严重的访问竞争而导致的性能瓶颈.我们测试了分区数对 NRPERTL 性能的影响,图 4 显示了在事务到达率固定为 40trans/s 情况下,分区数对系统性能的影响:*MDR* 随着分区数的增加而相应减小,但当分区数增加到 8 以后,再增加分区数,则对 *MDR* 无明显改进.这是因为分区数的增加同时也导致了系统故障后恢复处理代价的增加.

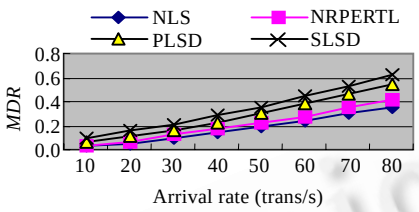


Fig.3 Comparison of four logging schemes

图 3 4 种日志模式的比较

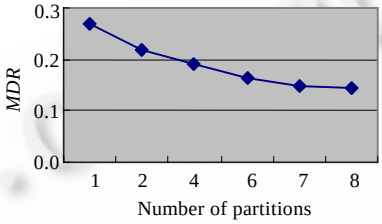


Fig.4 Influence of number of partitions

图 4 分区数对性能的影响

故障发生后,系统停止服务的时间T_{DT}是衡量恢复性能的另一项重要指标.对于DRTMMDBS而言,T_{DT}主要包括:1) 将数据从LSDB装入内存重建LMDB所需的时间T_1;2) 将LMDB恢复到一致性状态所需的时间T_2.由于RTDCRS采用了基于分类恢复思想的动态恢复策略(关键类数据被首先装入和恢复,在一般类数据装入内存前恢复了系统服务),因此对RTDCRS而言,T_{DT}等于将关键数据类装入内存所需时间与在内存中将它们恢复一致性状态所需时间之和.T_{DT}可估算如下:

$$T_{DT}=T_1+T_2\approx S_{LMDB}\times R_{CD}\times T_{I/O}+(SL\times R_{CL}\times\alpha)\div S_{LR}\times T_{rp} \quad (1)$$

式(1)中,$T_{I/O}$表示从外存读入一数据块到内存平均所需时间;R_{CL}表示关键日志区占整个可用日志区的比率;S_{LR}代表一条日志记录的平均大小;T_{rp}代表恢复时一条日志记录的平均处理时间;S_{LMDB},R_{CD},SL及α的含义如表 1 所描述.由于关键数据类通常只占全部数据的一小部分,因此,相比于将全部数据装入、恢复后再提供系统服务的静态恢复策略而言,RTDCRS能够明显地降低系统停止服务的时间T_{DT}.

4 结束语

DRTMMDBS 通常使用在时间关键类应用中,对这类应用而言,故障发生后能够迅速而有效的恢复是至关重要的.本文系统而深入地研究了适合于 DRTMMDBS 的系统故障恢复方法,包括日志模式、检验点以及故障后的恢复处理策略.本文的主要贡献包括:

(1) 在深入分析 DRTMMDBS 的故障恢复需求的基础上,给出了 DRTMMDBS 的恢复正确性准则;

(2) 提出了一种集成了分区和短暂日志特性的、基于非易失性 RAM 的实时日志模式 NRPERTL.结合动态恢复处理策略,给出了确保一致性恢复的日志分区原则与事务类分划的方法;

(3) 基于 NRPERTL 给出了与之相适应的本地模糊检验点模式;

(4) 在故障发生后的处理策略上,提出了基于分类恢复思想的动态恢复策略;

(5) 证明了上述恢复方法的正确性,并通过性能测试与评估验证了其性能的优越性.

尽管上述恢复方法主要是针对 DRTMMDBS 需求而提出来的,但其思想同样可以应用在传统的分布式数据库和集中式数据库的恢复处理中,以提高恢复效率与系统性能.

随着移动通信技术的迅速发展,移动计算环境下的应用有着不断增长的需求.在本文研究基础上,继续研究在移动计算环境下实时数据库恢复策略将是我们下一步的研究方向.

References:

[1] Liu YS. Advanced Database Technology. Beijing: Publishing House of National Defense Industry, 2001 (in Chinese).

[2] Choi MS, Yoon HS, Song EM, Kim YK, Jin S, Han M, Choi W. Two-Step backup mechanism for real-time main memory database recovery. In: Proc. of the 7th Int'l Conf. on Real-time Computing Systems and Applications. 2000. 453−457. http://ieeexplore.ieee.org/iel5/7188/19358/00896425.pdf

[3] Sivasankaran RM, Ramamritham K, Stankovic JA, Towsley D. Data placement, logging and recovery in real-time active databases. In: Berndtsson M, Hansson J, eds. Active and Real-Time Database Systems. London: Springer-Verlag, 1995. 226−241.

[4] John SK, William JD. Performance evaluation of ephemeral logging. In: Peter B, Sushil J, eds. Proc. of the '93 ACM SIGMOD Int'l Conf. on Management of Data. Washington: ACM Press, 1993. 187−196.

[5] Lam KY, Kuo TW. Real-Time Database Systems Architecture and Techniques. Boston: Kluwer Academic Publishers, 2001.

[6] Liu P, Ammann P, Jajodia S. Rewriting histories: Recovering from malicious transactions. Distributed and Parallel Databases, 2000, 8(1):7−40.

[7] Panda B, Tripathy S. Data dependency based logging for defensive information warfare. In: Proc. of the 2000 ACM Symp. on Applied Computing. 2000. 361−365.

[8] Song EM, Kim YK, Ryu C. No-Log recovery mechanism using stable memory for real-time main memory database systems. In: Proc. of the 6th Int'l Conf. on Real-Time Computing Systems and Applications. 1999. 428−431. http://ieeexplore.ieee.org/iel5/6590/17590/00811294.pdf

[9] Shu LC, Sun HM, Kuo TW. Shadowing-Based crash recovery schemes for real-time database systems. In: Proc. of the 11th Euromicro Conf. on Real-Time Systems. 1999. 260−267. http://ieeexplore.ieee.org/iel5/6302/16859/00777473.pdf

[10] Abraham S, Henry FK. Database System Concepts. 3rd ed., Beijing: China Machine Press/McGraw-Hill, 1999.

[11] Xiao YY. Researches on recovery and secure consistency of distributed real-time database [Ph.D. Thesis]. Wuhan: Huazhong University of Science and Technology, 2005 (in Chinese with English abstract).

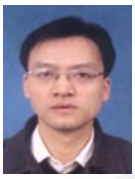
[12] Xiao YY, Liu YS, Liao GQ, Liu XF. Real-Time commitment in one-phase for distributed real-time transactions. Journal of Huazhong University of Science and Technology (Nature Science Edition), 2006,34(3):1−4 (in Chinese with English abstract).

附中文参考文献:

[1] 刘云生.现代数据库技术.北京:国防工业出版社,2001.

[11] 肖迎元.分布式实时数据库恢复与安全一致性研究[博士学位论文].武汉:华中科技大学,2005.

[12] 肖迎元,刘云生,廖国琼,刘小锋.分布式实时事务一阶段实时提交.华中科技大学学报(自然科学版),2006,34(3):1−4.



肖迎元(1969－),男,湖南邵阳人,博士,副教授,CCF 高级会员,主要研究领域为现代数据库技术,数据库与信息系统开发,实时信息处理.

廖国琼(1969－),男,博士,研究员,主要研究领域为现代数据库技术.

刘云生(1940－),男,教授,博士生导师,主要研究领域为现代数据库理论与实现,软件方法学与支撑环境.

邓华锋(1974－),男,博士生,主要研究领域为实时信息处理,流数据处理.