# 嵌入式实时内存数据库故障恢复技术*)

肖迎元　刘云生　刘小峰　廖国琼
(华中科技大学计算机学院　武汉 430074)

**摘　要**　故障发生后,迅速而有效地恢复对数据库系统而言是至关重要的。本文针对嵌入式实时内存数据库的特征,结合一个具体的嵌入式实时内存数据库系统(ARTs-EDB),给出了相应基于日志的恢复策略和实现技术。

**关键词**　嵌入式实时内存数据库,恢复,检验点,日志模式

## A Crash Recovery Technique for Embedded Real-Time Main Memory Databases

XIAO Ying-Yuan　LIU Yun-Sheng　LIU Xiao-Feng　LIAO Guo-Qiong
(School of Computer Science and Technology, HUST, Wuhan 430074)

**Abstract**　The rapid and efficient recovery in the event of failures is very important for database system. To aim at the characteristics of embedded real-time main memory database, this paper presents the logging-based recovery strategy and realization technique based on the material embedded real-time main memory database system (ARTs-EDB).

**Keywords**　Embedded, Real-time, Main memory database, Recovery, Checkpoint, Log scheme

嵌入式实时内存数据库(ERTMMDB)集成了嵌入式、实时数据处理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,因而有一系列的问题需要研究解决,其中故障恢复技术就是一个重要的、值得研究的课题。

嵌入式应用环境(ERTMMDB通常嵌入在PDA、智能卡等设备中使用)对ERTMMDB的稳定性和可靠性提出了更高的要求。实时应用特征(事务和数据有定时限制)则要求ERTMMDB在故障发生后能迅速地恢复,以最大限度地满足事务和数据的时限要求。由于采用了内存数据库体系结构,更使得ERTMMDB在恢复活动,包括检验点、重启动等方面与基于磁盘的数据库有较大的差异[1]。而内存的易失性和脆弱性使得ERTMMDB发生系统故障的概率更大[2]。另外在与恢复直接相关的事务调度策略、并发控制协议方面,ERTMMDB与传统的数据库系统也有了很大的不同,如传统的数据库采用的是先来先服务的事务调度策略,而ERTMMDB通常采用基于优先级可抢占的调度策略。在并发控制方面,传统的数据库普遍采用两段锁(2PL)协议,而ERTMMDB通常采用的是高优先级两段锁(HP-2PL)。上述这些,决定了不能将传统数据库的恢复策略和技术不加改造地直接应用到ERTMMDB上。本文基于我们自行开发的一个嵌入式实时内存数据库管理系统(下文简称ARTs-EDB),给出了基于日志的故障恢复模式和实现技术。

## 1　ARTs-EDB系统模型

在ARTs-EDB中,数据库包含内存"工作版本"(内存数据库:MMDB)和外存备份版本(外存数据库:SDB),内存数据库机制确保事务在执行过程中无磁盘I/O操作。ARTs-EDB包括如下主要部件:一个事务管理器(TM)、一个调度器(Scheduler)、一个并发控制管理器(CCM)、一个内存数据库管理器(MMDBM)、一个存储管理器(SM)和一个恢复管理器(RM)。TM负责对事务进行预处理(如分配唯一的事务标识、分派优先级等)并监控事务的提交和夭折。Scheduler依据事务的优先级实施事务调度。并发控制管理器负责控制并发事务的读、写操作的执行顺序,确保数据内部和时态的一致性。ARTs-EDB中采用了高优先级严格两段锁协议(HP-S2PL),事务在完成提交活动且更新被写入MMDB后,才释放所持有的锁。当高优先级事务与低优先级事务发生读写或写写冲突时,高优先级事务夭折低优先级事务获得相应的锁资源。MMDBM负责内存数据库的建立和管理。SM负责SDB的读写操作及相关维护工作。内外存数据交换的基本单位为固定大小磁盘块。RM负责日志的创建、检验点的执行及故障后系统的恢复。

当一个事务开始执行,首先申请一定数量的数据缓冲页(称为事务私有数据缓冲区)。读操作read(X)表示数据对象X从MMDB被读入相应事务私有数据缓冲区。写操作write(X)表示数据对象X被写入到对应事务私有数据缓冲区。一个事务写入或更新的私有数据缓冲区,在该事务提交后才被写入MMDB,即采用了"延迟写"技术。当执行检验点时,MMDB中所有更新页被刷新到SDB。

## 2　日志和检验点模式

### 2.1　日志执行模型

对于传统的顺序日志模式,并发事务的日志记录按对应事务操作的执行顺序依次记录在当前日志缓冲页中,当事务提交或日志缓冲区满时,相应日志记录被写入外存日志文件[3]。显然,对基于优先级可抢占调度策略的实时系统而言,当前日志缓冲页会因为严重的访问竞争而成为系统性能的瓶颈。

*)本文得到国家十五国防预研重点项目(413150403),国防预研基金项目(51415030203JW05)及中国博士后科学基金项目(2003034482)的资助。肖迎元　博士研究生,研究兴趣为分布式、实时、移动、内存等现代数据库技术。刘云生　教授,博士生导师,主要研究方向为现代(非传统)数据库理论与技术,软件方法学与支撑环境。

在我们日志模式中，为每一事务单独地分配一定数量临时日志缓冲页(简称为活动日志区)。如图1所示，对每一事务操作(包括读操作、写操作、事务提交、事务夭折)，日志管理模块(Logger)负责在该事务的活动日志区创建一条对应的日志记录。当一个事务的提交日志记录被写入其活动日志区后，活动日志区的内容才被刷新到外存日志文件。只有当一个事务的活动日志区的内容被刷新到外存日志文件后，其私有数据缓冲区的更新才能被写入到MMDB(遵循先写日志协议)。

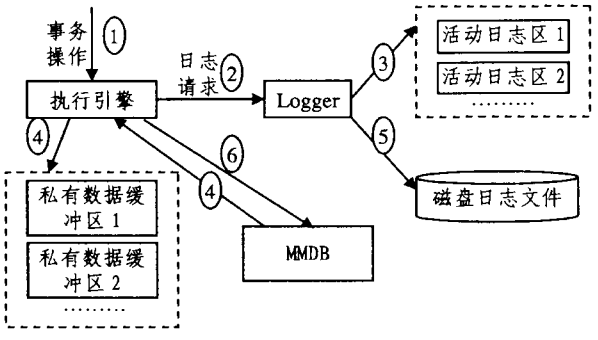


图1 日志执行模型

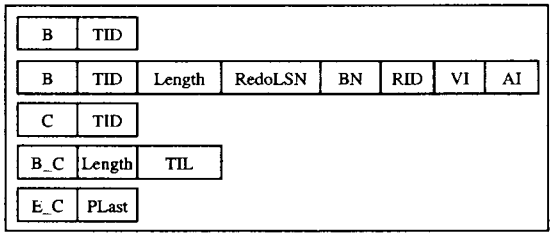


图2 五类日志记录的逻辑结构

### 2.2 日志记录类型和结构

由于采用了"延迟写"技术，一个事务的更新在提交后才被写入到MMDB，因而Undo日志记录不再需要。按照我们的日志执行模型，外存日志文件中只包含了五种日志记录类型：Begin，Redo，Commit，Begin_Checkpoint和End_Checkpoint。图2显示了五种日志记录类型的逻辑结构。图中，B，R，C，B_C和E_C分别表示Begin，Redo，Commit，Begin_Checkpoint和End_Checkpoint五种日志记录类型；TID为该日志记录所属事务的标识；RedoLSN为Redo日志记录顺序号，其值按Redo日志记录创建的顺序单调递增；Length用来表示Redo或E_C日志记录的长度；RID表示被更新的数据对象的标识；BN表示与MMDB中数据页P相对应的外存数据块(包含在SDB中)的逻辑块号，这里P表示被更新数据对象所在的内存页；VI表示时态数据对象的有效时刻，即VI::= ST+VP，这里，ST表示时态数据对象的采样时刻，VP表示时态数据对象有效期。对于非时态数据，VI被置为-1；AI表示数据对象的更新值(后映像)；TIL(事务信息表)用来记录Begin_Checkpoint日志记录创建时系统中事务的相关信息。事务信息表的结构在下文介绍；PLast指向上一个Begin_Checkpoint日志记录在日志文件中的位置。

### 2.3 主要数据结构

根据故障后恢复的需要，我们将事务执行的不同阶段划分成五种不同的状态：a. 活跃状态(active)，已开始执行但提交尚未开始；b. 提交状态(committing)，处在将活动日志区刷新到外存日志文件过程中；c. 提交后状态(committed)，提交已完成即Commit日志记录已被写入外存日志文件，但更新还没完全反映到MMDB中；d. 夭折状态(aborted)，事务因为锁冲突或自身逻辑错误被终止；e. 完成状态(finished)，事务的更新已完全刷新到MMDB中。

为了恢复的需要，系统在内存中创建并维护两个表：事务信息表(TIL)和MMDB页表。TIL由事务管理器创建并维护，为进入系统的每一事务创建并维护一条记录，每一条记录包含三个域：TID，LastRedoLSN和State。TID表示事务标识；LastRedoLSN表示该事务最近的Redo日志记录的RedoLSN的值；State用来记录事务所处的状态，State的值随着事务状态的变化被动态更新。当一个事务进入完成状态或夭折状态后，TIL中对应该事务的记录被删除。即TIL中只记录了处在活跃状态、提交状态和提交后状态的事务信息。MMDB页表为MMDB中每一数据页维护一项，每一项包含四个域：PID，BID，Ubit和CurRedoLSN。其中，PID为MMDB中数据页标识(用来确定该页在MMDB中的位置)；BID表示数据页PID在SDB中所对应的数据块的逻辑块号；Ubit为更新位，用来表示MMDB数据页PID的更新是否已被反映到SDB中，Ubit为1，表示数据页的更新还未被刷新到SDB。在后面的叙述中，我们将MMDB中Ubit位为1的数据页简称为脏页。Ubit为0表示更新已被刷新到SDB，PID和对应BID处在一致性状态；CurRedoLSN表示对数据页进行的最近的一次更新操作所产生的Redo日志记录的RedoLSN的值。同样，对SDB中每一数据块，也在块头的控制信息中设置一个CurRedoLSN域，用来保存该块最近一次更新操作对应的Redo日志记录的RedoLSN的值。MMDB页表由恢复管理器负责维护。

### 2.4 检验点模式

由于检验点的执行涉及到大量的磁盘I/O操作，故检验点模式(静态或动态)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MMDB的更新刷新到SDB的过程中，系统中不允许存在并发的事务，显然这不利于满足实时数据库系统中事务和数据的时限要求。ARTs-EDB采用动态检验点模式，检验点进程(线程)和事务并发执行。在检验点触发时机上，ARTs-EDB不是采用周期性触发策略，而是根据MMDB页表中脏页所占比率(简记为$R_{DP}$)是否到达设定的阈值$\alpha$来决定是否触发检验点进程。当$R_{DP}$大于$\alpha$，检验点进程被触发。$\alpha$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通过修改$\alpha$的值来动态调整检验点进程的执行频率。

检验点进程的执行过程描述如下：

(1)读取事务信息表的内容，写Begin_Checkpoint日志记录到外存日志文件；

(2)根据MMDB页表依次将脏页刷新到SDB，同时修改相应的Ubit位(将Ubit位置0)。在具体刷新每一脏页到SDB时，采用互斥量以确保该页在刷新过程中不会被并发事务所修改；

(3)写End_Checkpoint日志记录到外存日志文件，并在日志文件的头结构(存放控制信息)的LastChPointer域中记录下本次Begin_Checkpoint日志记录在日志文件中的位置。

由于检验点进程和事务并发执行，为了确保检验点进程能尽快完成，我们为检验点进程设置较高的优先级。

## 3 故障恢复

故障发生后，迅速而有效地恢复对嵌入式实时内存数据库而言是至关重要的[4]。对事务故障(事务夭折)，由于采用了延时写技术，事务对数据库的更新在事务提交后才能真正

写入数据库。因而,对这类故障只需简单地释放事务的私有数据缓冲区和对应的活动日志区即可,无需对数据库执行部分 Undo 操作。对于系统故障,它导致系统非正常终止,造成易失性主存数据丢失(MMDB 遭到破坏),从而导致一些新近提交的事务(它们对数据库的更新可能已经写入 MMDB,但尚未刷新到或尚未完全刷新到 SDB)的效果丢失。系统重启后,这部分事务必须被 Redo,以确保事务的持久性(Durability)。本文的故障恢复算法主要针对于系统故障。

系统故障发生后,恢复处理过程主要分为二个阶段:分析阶段和 Redo 阶段。

### 3.1 分析阶段

分析阶段通过分析日志来确定所有需要重做(Redo)的事务并确定 Redo 起始点。如图 3 所示,LS_Checkpoint 表示在故障发生前最后一次成功完成的检验点;T1 表示在 LS_Checkpoint 的 Begin_Checkpoint 记录写入日志文件以前活动日志区已刷新到日志文件的事务(即包括在 Begin_Checkpoint 记录写入前已处在提交后状态和完成状态的事务);T2 表示部分日志记录在 Begin_Checkpoint 之前写入,部分日志记录在 Begin_Checkpoint 之后写入的事务;T3 表示日志记录在 Begin_Checkpoint 之后,但在 End_Checkpoint 之前写入日志文件的事务;T4 表示部分日志记录在 End_Checkpoint 之前写入,部分在 End_Checkpoint 之后写入的事务;T5 表示日志记录在 End_Checkpoint 之后但在故障发生前写入日志文件的事务;T6 表示日志记录在写入日志文件过程中发生系统故障的事务(部分日志记录已写入日志文件,但 Commit 日志记录未被写入的事务)。对于 T2, T3, T4, T5 这四类事务,由于 Commit 日志记录在故障发生前已写入日志文件,但这四类事务对数据库所做更新在故障发生前不一定真正写入 SDB,因此恢复时必须进行 Redo。T6 类事务由于其 Commit 日志记录尚未写入日志文件,因此无需 Redo。对于 T1 类事务,在基于先来先服务调度策略的传统数据库系统中,由于所有日志记录在 LS_Checkpoint 开始之前已写入日志文件,按照先来先服务调度策略,它们对数据库的更新也应该在 LS_Checkpoint 执行前被写入到 MMDB。因而在故障发生时,更新已被写入 SDB,所以对于传统数据库系统而言,T1 类事务无需 Redo。然后对于嵌入式实时数据库系统而言,由于采用了基于优先级可抢占的调度策略,事务在将活动日志区刷新到日志文件后,更新尚未写入或尚未全部写入 MMDB 时,此时 CPU 可能被其它高优先级事务或正好被触发的 LS_Checkpoint 进程抢占,从而导致事务对数据库更新未能在 LS_Checkpoint 过程中完整地刷新到 SDB,当故障发生,导致事务效果丢失。我们将 T1 类事务进一步划分为两类:T11 和 T12。T11 表示在 Begin_Checkpoint 记录写入日志文件以前已处在完成状态的事务;T12 表示在 Begin_Checkpoint 记录写入日志文件时还处在提交后状态的事务。显然 T11 类事务无需 Redo,而 T12 类事务必须 Redo。分析阶段主要工作是确定 T2 类事务的集合 $S_{T2}$ 及 T 12类事务的集合 $S_{T12}$ 并在日志文件中找到 $S_{T2} \cup S_{T12}$ 中事务的所有 Begin 日志记录中最早的(重做起始点)。具体的处理步骤如下:

(1)置 $S_{T2}=\phi$;

(2)从日志文件尾开始反向扫描,直到 LS_Checkpoint 的 Begin_Checkpoint 记录为止;

(2.1)对每一 Commit 日志记录,令 $S_{T2} = S_{T2} \cup \{TID\}$,其中,TID 为该 Commit 日志记录所属事务的标识;

(2.2)对每一 Begin 日志记录,令 $S_{T2} = S_{T2} - \{TID\}$,其中,TID 为该 Begin 日志记录所属事务的标识;

(3)根据 LS_Checkpoint 的 Begin_Checkpoint 日志记录的 TIL 域确定集合 $S_{T2}$,即 $S_{T2} = \{TID \mid \exists R \in TIL,\ TID = R.TID,\ R.State = committed\}$,这里,R 表示 TIL 中的一条记录。R. TID 和 R. State 分别表示 R 的 TID 域和 State 域的值;

(4)令 $S = S_{T12} \cup S_{T2}$;

(5)从 LS_Checkpoint 的 Begin_checkpoint 日志记录处反向扫描日志文件,直到 $S=\varphi$;

(5.1)对每一 Begin 日志记录,令 $S=S-\{TID\}$;

(6)将当前 Begin 日志记录的位置记录下来作为 Redo 起始点。

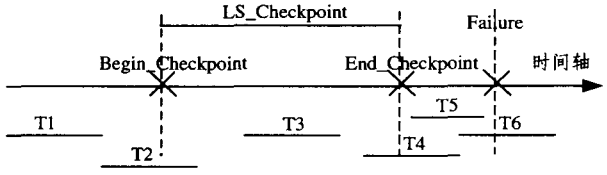


图 3 日志记录时序图

### 3.2 Redo 阶段

Redo 阶段依据分析阶段确定的 Redo 起始点,执行重做操作,将 SDB 恢复到最近的一致性状态。为了降低 I / O 操作的代价,我们的恢复策略不是在 Redo 阶段完成后,再进行 MMDB 的装入、重建,而是在执行 Redo 的同时重建 MMDB。由于时态数据超过了有效期后,它的值就变得无意义(这些失效的时态数据通过下一个采样周期更新事务的写入而得到更新)[5],因此,在 Redo 阶段,对失效的时态数据并不执行 Redo 操作。Redo 阶段具体的实现步骤描述如下:

(1)从 Redo 起始点开始,正向扫描日志文件,直到日志文件尾;

(1.1)对于每一 Redo 日志记录 $R_T$,假定其所属事务为 T,执行下面操作;

(1.1.1)IF(T 的 Commit 日志记录存在且 $R_T$. VI 大于系统当前时间)

将外存 SDB 中逻辑块号为 RT. BN 的块读入内存,装入 MMDB 对应数据页;

(1.1.1.1)IF(该块的 CurRedoLSN 的值小于 $R_T$. RedoLSN);

用 $R_T$. AI 覆盖 MMDB 中对应数据页的相应数据对象,并将修改后的数据页刷新到 SDB。

在上面的描述中,$R_T$. VI 表示 Redo 日志记录 $R_T$ 的 VI 域的值;$R_T$. RedoLSN 表示 $R_T$ 的 RedoLSN 域的值。

Redo 阶段实现了将 SDB 恢复到最近的一致性状态,同时重建了与 SDB 一致的内存工作版本(MMDB),当然此时 MMDB 只是 SDB 的部分映像。若此时 MMDB 的剩余空间占整个 MMDB 空间的比率仍然超过预先设定的重装阈值 β,则按照故障重装策略继续装入 SDB 中其它数据块,直到超过重装阈值 β,然后恢复系统服务。

**结束语** 嵌入式应用环境、实时应用特征及内存数据库的体系结构使得嵌入式实时内存数据库对故障恢复技术提出

(下转第 105 页)

丢弃20%的元组，即这个区间的一半数据。被丢弃的区间为[0,25)，插入的语义丢弃操作符谓词应为值≥25。如果需要丢弃70%的元组，则区间[0,50]将不能满足要求。此时，需要丢弃区间[0,50]的全部元组以及区间[51,100]元组的一半。插入的语义丢弃操作符谓词应为值75。

**结论和展望** 对计算机系统来说，能够适应操作环境的变化是非常重要的。特别是对监测连续的数据流，这一点显得尤为重要，因为数据流中的数据的速度和特征是不可预测地发生着变化。当数据的到达速度超出系统的处理能力时，需要卸载来保证系统正常运行。常用的卸载方法是在查询计划当中插入随机卸载操作符或语义卸载操作符。卸载时需要解决的关键问题是何时、何处卸载以及丢掉多少负载。卸载的目标可以是使系统的输出速度最大，或者是对答案的精确度影响最小等。研究的卸载问题涉及到聚合查询、连接查询等。

对数据流中卸载技术的深入研究工作包括：(1)对低负载系统的卸载处理。当数据速度太低时，元组也应该丢弃掉(超时)。下一步的工作应该包括研究当发生高负载与低负载时，进行卸载有哪些相同和不同；(2)如何将卸载算法扩展到其它的资源管理；(3)语义卸载的处理还应进一步完善。

## 参 考 文 献


1 The Stanford Stream Data Manager. http://www-db.stanford.edu/stream
2 Carney D, Cetintemel U, Cherniack M, et al. Monitoring streams—a new class of data management applications. In: Proc. Int. Conf. on Very Large Databases (VLDB), 2002
3 Chandrasekaran S, Deshpande A, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR, Jan. 2003
4 Chen J, DeWitt D J, Tian F, et al. NiagaraCQ: A scalable continuous query system for internet databases. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000. 379～390
5 Floyd S, Paxson V. Wide-area traffic: The failure of poisson modeling. IEEE/ACM Transactions on Networking, 1995, 3(3): 226～244
6 Kleinberg J. Bursty and hierarchical structure in streams. In: Proc. 2002 ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, Aug. 2002
7 Leland W, Taqqu M, Willinger W, Wilson D. On the self-similar nature of ethernet traffic. IEEE/ACM Transactions on Networking, 1994, 2(1): 1～15
8 Tatbul N, Cetintemel U, et al. Load Shedding in a DataStream Manager. VLDB 2003
9 Ayad A M, Naughton J F. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. SIGMOD 2004
10 Babcock B, Datar M, Motwani R. Load Shedding for Aggregation Queries over Data Streams. ICDE 2004
11 Hoeffding W. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 1963, 58: 13～30
12 Motwani R, Widom J, Arasu A, et al. Query processing, approximation, and resource management in a data stream management system. In: Proc. First Biennial Conf. on Innovative Data Systems Research (CIDR), Jan. 2003
13 Yang C, Reddy A V S. A Taxonomy for Congestion Control Algorithms in Packet Switching Networks. IEEE network, 1995, 9(5): 34～44
14 Compton C L, Tennenhouse D L. Collaborative Load Shedding for Media-Based Applications. In: Intl. Conf. on Multimedia Computing and Systems, Boston, MA, May 1994. 496～501
15 Zhu Y, Shasha D. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In: Proc. Int. Conf. on Very Large Data Bases, 2002. 358～369
16 Das A, Gehrke J, Riedwald M. Approximate join processing over data streams. In: Proc. 2003 ACM SIGMOD Conf., 2003. 40～51
17 Kang J, Naughton J F, Viglas S. Evaluating window joins over unbounded streams. In: Proc. 2003 Intl. Conf. on Data Engineering, Mar. 2003
18 Liu L, Pu C, Tang W. Continual Queries for Internet-Scale Event-Driven Information Delivery. IEEE Trans. Knowledge and Data Eng., 1999, 11(4): 610～628
19 Abadi D, Carney D, Cetintemel U, et al. Aurora: A Data stream Management System. In: ACM SIGMOD conference, San Diego, CA, June 2003. 666
20 Abadi D, Carney D, Cetintemel U, et al. Aurora: A New Model and Architecture for Data Stream Management. To be appeared in VLDB Journal
21 Tatbul N, Cetintemel U, Zdonik S, et al. Load Shedding in a Data Stream Manager: [Technical Report CS-03-03]. Brown University, Computer Science, Feb. 2003
22 Babcock B, Datar M, Motwani R. Load shedding techniques for data stream systems (full version). Draft in preparation, 2003


(上接第79页)

了新的要求。本文结合一个具体的嵌入式实时内存数据库系统，提出了一个适合于嵌入式实时内存数据库的基于日志的故障恢复模式并给出了相应数据结构和实现算法。不同于传统的顺序日志模式，在我们的恢复模式中，为并发的每一事务分配独立的活动日志区，不同事务的日志记录被临时记录在各自的不同活动日志区，从而解决了顺序日志模式中当前日志缓冲页因为严重的访问竞争而影响系统性能的问题；在检验点模式上，我们采用了执行频率动态可调的动态检验点模式，提高了并发度，很好地解决了如何确定检验点触发时机和执行频率的问题。系统故障发生后，在恢复处理过程中，就Redo起始点的选择做了仔细分析，确保提交事务持久性的满足。在恢复SDB的同时，重建了内存数据库(MMDB)，从而降低了故障恢复时间。在恢复的具体实现算法中，充分考虑了数据和事务的时限要求，尽可能地减少内外存I/O操作的次数，从而有利于实时事务截止期的满足。

本文提出的故障恢复模式也存在一定局限性，如基于活动日志区的日志执行模型的正确性是以严格两段锁协议(并发控制协议)作为前提。进一步的改进在于：采用更具一般性的分区日志技术(放宽对并发控制协议的要求)；采用非易失高速存储设备作为日志存储区；采用多日志文件代替单一日志文件等。这些技术的使用可进一步改进恢复的效率，但同时也可能增加实现的复杂度。在后续的工作中，我们将在恢复效率和实现复杂度方面进行权衡，进一步改进我们的恢复策略。

## 参 考 文 献


1 胡国玲，刘云生，彭嘉雄. 内存数据库系统的恢复技术. 华中理工大学学报，1996，24(3)：35～38
2 Eliezer L, Avi S. Incremental Recovery in Main Memory Database Systems. IEEE Transactions on Knowledge and data Engineering, 1992, 4(6): 529～540
3 Lam K Y, Kuo T W. Real-Time Database Architecture and Techniques. Boston: Kluwer academic publishers, 2001
4 Huang J. Recovery Techniques in Real-Time Main Memory Databases: [Ph. D. Dissertation]. School of Computer Science, The University of Oklahoma
5 刘云生. 现代数据库技术. 北京：国防工业出版社，2001