第32卷 第11期 计 算 机 学 报 Vol. 32 No. 11
2009年11月 CHINESE JOURNAL OF COMPUTERS Nov. 2009

# 面向更新密集型应用的内存数据库高效检查点技术

覃雄派[1),2)] 肖艳芹[1),2)] 曹 巍[1),2)] 王 珊[1),2)]

[1)](教育部数据工程与知识工程重点实验室(中国人民大学) 北京 100872)

[2)](中国人民大学信息学院 北京 100872)

**摘 要** 面向更新密集型应用的内存数据库系统,其检查点技术应符合几个关键的要求,包括检查点操作对正常事务处理的干扰尽可能小、能够处理存取倾斜状况、支持数据库系统的快速恢复、提供恢复过程中的系统可用性等.该文提出一种事务一致的分区检查点技术,采用基于元组的动态多版本并发控制机制,避免了读写事务的加锁冲突,提高系统吞吐能力;检查点操作以只读事务形式实现,在多版本并发控制下,避免检查点操作对正常事务处理的堵塞;由于检查点文件是事务一致的,只需要记录事务的Redo日志信息,在系统恢复过程中,只需要对日志文件进行一遍扫描处理,加快恢复过程;基于优先级的数据分区装载和恢复,使得恢复过程中新事务的数据存取请求迅速得到满足,保证了恢复过程中的系统可用性.由于采用两级版本管理机制以及动态版本共享技术,多版本管理的空间开销降低到可以接受的水平.实验结果表明,文中提出的检查点技术方案获得比模糊检查点技术高27%的系统吞吐量,同时版本管理的空间开销在可接受的范围之内,满足高性能应用的要求.

**关键词** 内存数据库;动态多版本管理;事务一致检查点;元组级

**中图法分类号** TP311 **DOI号**: 10.3724/SP.J.1016.2009.02200

# An Efficient Checkpointing Scheme for Update Intensive Applications in Main Memory Database Systems

QIN Xiong-Pai[1),2)] XIAO Yan-Qin[1),2)] CAO Wei[1),2)] WANG Shan[1),2)]

[1)](*Key Laboratory of Data Engineering and Knowledge Engineering* (*Renmin University of China*), *MOE*, *Beijing* 100872)

[2)](*School of Information*, *Renmin University of China*, *Beijing* 100872)

**Abstract** Four requirements for checkpointing in update intensive main memory database systems are identified, namely how checkpointing interferes with normal transaction processing, how checkpointing handles skewed access pattern, whether checkpointing supports fast restart, and whether checkpointing provides system availability during recovery. A partition based transaction-consistent checkpoint scheme is proposed in this paper, the scheme uses a tuple-level dynamic multi-versioning concurrency control protocol to avoid the lock conflicts between read only transactions and update transactions, thus achieves high system throughput. Checkpointing is implemented as data partition read only transactions, and incurs little interfering with normal transaction processing, furthermore transaction-consistent checkpoint requires writing out only redo log records of committed transactions, during recovery only one pass of log scanning is needed. Priority-based on demand partition recovery gives high attention to data access demands of executing transactions, provides system availability during recovery, which is critical for high performance applications. Finally, two level version management and dynamic version sharing re-

---

收稿日期:2008-08-02;最终修改稿收到日期:2009-06-16.本课题得到国家自然科学基金(60496325,60493069)和国际合作项目基金资助.**覃雄派**,男,1971年生,讲师,博士研究生,主要研究方向为数据库查询优化、高性能内存数据库. E-mail: qxp1990@sina.com. **肖艳芹**,女,1974年生,博士研究生,主要研究方向为内存OLAP数据库. **曹 巍**,女,1975年生,讲师,博士研究生,主要研究方向为数据库优化、自管理自调优数据库. **王 珊**,女,1944年生,教授,博士生导师,主要研究领域为高性能数据库新技术,包括内存数据库、数据仓库与知识工程、数据库信息检索等.

duce space overhead to an acceptable level. Experiment results demonstrate effectiveness and efficiency of the proposed scheme.

**Keywords** main memory database systems; dynamic multi-versioning; transaction-consistent checkpoint; tuple level

# 1 引 言

内存数据库的所有数据长驻内存,在事务处理的过程中,避免了 IO 操作,系统获得极高的性能.为了保证事务的 ACID 特性,在事务提交的时候,必须进行日志的记录,需要进行 IO 操作.内存数据库由于其巨大的吞吐能力,在更新事务密集型应用(如电信业务应用)中,生成的日志数量是惊人的,必须进行检查点操作以便截短日志文件.同时,内存的脆弱性,也要求不时地把内存影像保存到磁盘等可靠的存储器上.

面向更新密集型应用的内存数据库系统,其检查点技术的优劣,必须从以下几个方面来衡量.①检查点操作对正常事务处理的干扰程度.检查点操作对正常事务处理的干扰越小,越能够发挥内存数据库系统的事务处理能力,不会因为进行检查点操作而造成正常事务处理的堵塞,使内存数据库系统获得更高的吞吐能力.②对存取倾斜状况的处理能力.数据库的负载一般具有存取倾斜的特点,如80%的存取指向20%的数据,检查点操作必须能够处理这种状况,根据数据的更新频率确定其检查点操作频率,减少恢复过程所需要扫描的日志数量.③是否支持数据库系统的快速恢复.内存数据库系统是为高性能应用而设计的,在系统失败之后,恢复子系统必须把数据库迅速恢复到一致状态,全力投入新的事务处理工作当中.④系统在恢复过程中是否具有可用性(System Availability during Recovery).在恢复过程中,如果新事务所需要的数据率先得到装载和恢复,新事务就可以和恢复过程并发执行,无需等到整个数据库恢复完毕.内存的容量越来越大,TB 级的内存数据库系统已经成为现实,对如此大规模的内存数据库进行恢复,需要的时间以小时计[①].作为 McObject 性能测试工作的一部分,McObject 的工程师用 4.3h 完成了内存数据库的备份,而恢复则需要 4.76h(数据库大小为 1.17TB).如果在恢复的过程中系统不能够处理新的事务,新事务必须等待恢复过程结束才能得到处理,对于高性能应用来讲,是不可以忍受的.虽然内存的容量不断增加,然而数据库数据量的增加更快,在内存容量不能容纳所有数据的情况下,通过保证热点数据常驻内存,由于数据库的大部分负载一般存取热点数据,可以充分利用内存快速存取的优势提高事务处理的效率.内存数据库采用面向内存快速存取的数据结构(使用指针)和索引(T-Tree 等),有别于磁盘数据库采用的数据结构(数据块和磁盘地址)和索引(B+Tree),对于不能完全装载到内存中的其它数据,可以通过操作系统的虚拟内存技术进行这部分数据的内外存交换[②],内存地址和外存地址的转换由虚拟内存技术解决;或者由内存数据库库系统直接进行非热点数据的内外存交换[1],并完成地址转换.

本文提出基于动态版本管理的事务一致的分区检查点技术方案 PB-TCC-DV(Partition-Based Transaction-Consistent Checkpoint on Dynamic Versioning).我们在磁盘数据库多版本两阶段锁协议[2]的基础上,提出面向内存数据库的动态多版本两阶段锁并发控制方法.通过多版本管理,避免了读写事务的加锁冲突,检查点操作以只读事务形式实现(读取提交数据,刷新检查点文件),减少检查点操作对正常事务处理的干扰,有利于系统获得更高的吞吐能力.由于采用事务一致的检查点技术,日志文件只需要记录提交事务的 Redo 日志,恢复过程中只需进行一遍日志扫描和应用,加快了恢复过程.PB-TCC-DV 采用元组级的版本管理,以及两级版本管理机制和动态版本共享技术,版本管理的空间开销减少到可以接受的水平.此外,该方案还提供了恢复过程中的系统可用性.

本文第 2 节介绍相关的工作;第 3 节详细讨论 PB-TCC-DV 分区检查点技术;第 4 节介绍实验结果;最后对全文进行总结.

---

① McObject LLC. In-memory database systems beyond the terabyte size boundary[EB/OL]. http://www.mcobject.com. 2007-12-1/2008-1-15

② Hnik K. FastDB: Main memory relational database management system[EB/OL]. http://www.fastdb.org/fastdb.html. 2008-12-20/2009-6-5

## 2 相关工作

在模糊检查点技术(fuzzy checkpointing)实现中,检查点操作和活动事务并发执行,活动事务对数据的修改被刷新到磁盘上,如果此后某些活动事务回滚或者退出,将导致检查点文件中包含脏数据(dirty data).此外,当事务提交的时候,为了减少IO操作,不要求把事务的更新立刻刷新到磁盘上(NO-FORCE),于是已经提交的事务,有可能没有反映在检查点文件中.

模糊检查点操作由于对正常事务处理的干扰很小,面向内存数据库的模糊检查点技术得到了广泛的研究和应用.为了提高检查点操作的效率,同时进一步降低检查点操作对正常事务处理的干扰和影响,保证正常事务处理的效率,基于数据分区的检查点技术得到了深入的研究[3-5].文献[3]提出动态分段(dynamic segmented)模糊检查点技术.数据库被划分成不同的片段,在每个检查点周期里只转储一个片段(segment).各个片段以Round-Robin的方式进行转储,片段的边界可以根据事务对各个片段的更新频率动态地调整.检查点操作一个片段紧接着一个片段持续进行.最近的 $N+1$ 个片段检查点集合构成整个数据库的全局检查点.相对于传统的模糊检查点技术来讲,系统失败之后,需要处理的日志记录减少,所以分段模糊检查点技术的恢复总时间比传统模糊检查点技术更短.文献[4-5]提出把整个数据库划分成一系列分区(partition),每个分区单独进行检查点操作,每个分区有专属的日志磁盘,使用多个日志磁盘提高恢复的效率.日志记录根据分区ID进行分组,写入该分区的日志磁盘.通过日志分组和多个磁盘的I/O操作,系统失败后的恢复时间得以缩短.

上述技术方案,由于使用模糊检查点技术,在数据库恢复过程中,需要对日志文件进行两趟扫描(Undo Pass和Redo Pass),对退出事务做撤销操作(这些事务的脏数据已经刷新到检查点文件中),对提交事务做重做操作(这些事务提交的时候,并没有强制刷盘,检查点文件中尚未包含这些事务的更新),才能把数据库恢复到一致的状态.

PB-TCC-DV检查点技术方案同样以数据分区作为检查点操作的基本单位,各个数据分区的检查点操作频率依据更新频率进行计算.但是各个数据分区的检查点操作是事务一致(transaction-consistent)的,一致检查点技术的主要优势是日志处理的简化,从而能够加快恢复过程.

一致检查点技术(consistent checkpointing),只把提交的数据刷新到磁盘上,检查点文件包含数据库的一个历史快照.为了把数据库恢复到最近一致状态,首先把检查点文件装载进来,并且重做检查点操作以来提交的事务即可(事务的回滚和退出在内存中完成,脏数据并未写入磁盘).日志文件只需要记录提交事务的redo日志信息,恢复过程中只需要一趟日志文件的扫描处理(redo pass),避免模糊检查点技术所需要的两趟日志扫描(undo pass和redo pass),节省了磁盘IO,从而加快内存数据库的恢复过程.

一致检查点技术有两种实现方式,分别是对数据进行加锁以获得数据的一致视图,和通过多版本管理获得一致试图.加锁方式由于对正常的事务处理造成了堵塞,所以不适用于内存数据库.随着内存容量的增大,基于版本管理的内存数据库一致检查点技术,重新受到了研究人员的重视.文献[6]讨论了基于元组更新拷贝(tuple level copy-on-update)的一致检查点技术.SIREN[6]系统采用一种逻辑页面结构,内存中的元组以链表的方式连接起来构成页面.这种组织方法有两个好处,一个是可以进行元组级别的copy-on-update操作(copy-on-update操作是建立一致检查点的基础),另外一个好处是可以重新在内存里面组织元组,保证磁盘写操作的高效率.SIREN通过索引结构里的元组延迟插入(pending tuple add)以及元组延迟删除(pending tuple remove),支持无堵塞的一致检查点操作.文献[6]提出的检查点技术不仅解决了一致检查点操作空间开销过大的问题,而且避免了检查点操作对正常事务处理的堵塞.其版本管理的空间开销为影子页面(shadow paging)技术的10%,系统总体性能则比基于模糊检查点技术的系统提高30%左右.但是,该方案未提供恢复过程中的系统可用性.

本文提出的PB-TCC-DV检查点技术方案,基于临时版本技术实现一致检查点操作.临时版本技术(transient versioning)是一种多版本管理技术,数据的旧版本被临时保存起来,以提高系统的并发度.文献[2,7]等对磁盘数据库的临时版本两阶段锁(transient version two phase locking)并发控制和版本管理协议进行了深入研究.文献[7]讨论了页面级的多版本管理技术,采用这样的调度办法,查询事务(只读事务)读取数据的提交版本获得结果,不需要对数据加锁,从而不会被堵塞,也没有造成对更新事务的加锁冲突.数据库系统维护一个版本池

(version pool),用于保存数据页的旧版本,以支持查询事务.对版本池的写入是顺序写的方式,有利于充分利用磁盘的特点加速 IO 操作,空间回收算法采用先进先出(FIFO)的策略进行版本回收,当数据的旧版本没有任何查询事务需要的时候,即可进行清除和回收.但是先进先出的版本回收机制导致了一个严重问题,即如果有一个查询事务执行时间比较长(查询的数据比较多),则可能导致版本池积累大量的数据不能及时回收.面向磁盘的数据库系统,由于数据版本存储在磁盘上,长查询事务导致版本池膨胀,从而导致查询事务的抖动[2].Bober 和 Carey[2]对上述算法进行改进,支持元组级的多版本管理.每个页面开辟一块区域,保存元组的旧版本.页面内部的版本缓冲,有利于旧的元组版本尽快地得到重复利用,减少了对版本池(version pool)的 IO 操作;另外,数据的旧版本保存在数据页面上,有利于提高磁盘 IO 的效率,即通过一次 IO 操作,完成页面数据和临时版本的提取.在 PB-TCC-DV 检查点技术方案中,版本缓存的主要作用是加速版本重用,节省内存空间开销,而非减少 IO 操作(内存数据库不需要磁盘 IO 进行数据存取).版本缓存不隶属于数据页面,而是隶属于数据分区,版本缓存的大小是一个数据页面.由于元组在版本缓存(version cache)得到重用,减轻了版本在版本池的堆积,数据段共享的版本池(version pool)用以支持长查询事务.此外,PB-TCC-DV 通过动态版本共享技术进一步减少版本管理的空间开销.在这样的版本管理机制下,避免了使用两阶段锁协议(2PL)[8]进行事务调度的弊端,即一个长查询事务如果长时间持有锁,那么必然导致更新事务的等待.检查点事务(只读)以无堵塞的方式运行,减少对正常事务处理的干扰,使系统获得更高的吞吐量.

文献[9]提出用事务串行执行的办法来减少事务加锁的开销,其核心思想是,在内存数据库里,对于短事务的执行来讲,加锁的开销占用了大部分的系统资源,加锁等待时间过长;通过串行执行,可以减少加锁开销,提高事务执行效率.我们认为事务的串行执行不能充分利用多核处理器的计算能力,而多核处理器是 CPU 的发展趋势①,必须充分利用多个处理核心的计算能力提高事务处理的效率.

## 3　PB-TCC-DV 一致检查点技术

### 3.1　概　述

PB-TCC-DV 检查点技术方案,在动态多版本管理的基础上实现一致检查点操作.在动态多版本并发控制机制下,只读事务读取数据的提交版本(Latest Committed Version)获得结果,不需要对数据加锁,检查点操作实现为数据分区的只读事务,检查点操作和正常事务处理的耦合被解除,减少了检查点操作对正常事务处理的干扰,有利于系统获得更高的性能.多版本管理必然导致空间开销,PB-TCC-DV 一致检查点技术方案的本质是以空间换时间,即通过利用一部分内存进行版本管理,以换取检查点操作和事务执行效率的提高.试验证明,内存开销在可接受的范围之内,而系统性能的提高是可观的.检查点操作的频率根据数据分区的更新频率进行计算,保证数据库的更新被不断保存到检查点文件中,使得恢复过程需要扫描的日志总量减少,加快恢复过程.

### 3.2　一致检查点的基础:动态多版本并发控制

为了提高检查点操作的效率,我们对传统的段页式内存数据组织形式进行了改进,增加一个级别的数据管理层次——数据分区(data partition).一个数据库由多个段(segment)组成,每个段管理隶属于一个关系的数据和索引,段由若干个数据分区(partition)组成,每个数据分区由若干个定长的页面(page)组成(图 1).每个记录有一个唯一的标识 RID,RID 是一个四元组〈Segment ♯,Partition ♯,Page ♯,Slot ♯〉,分别表示段号、分区号、页面号和页内的记录槽号,记录槽(record slot)包含对应记录的首地址和记录的长度.

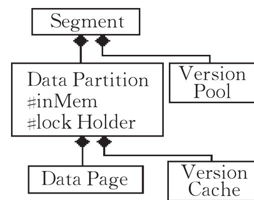



图 1　存储引擎的层次结构

我们在磁盘数据库多版本两阶段锁并发控制协议[2]的基础上,提出面向内存数据库的动态版本两阶段锁并发控制协议,进行事务的调度执行.主要的改进包括:(1) 以数据分区为加锁粒度,以保证系统的并发度,同时避免过大的加锁开销;(2) 通过两级版本管理避免版本堆积和加快版本回收;(3) 通过动态版本共享进一步减少版本管理空间开销,为高效的一致检查点操作做好准备.

① Intel Corporation. Intel Quad Core Xeon Processor Datasheet [EB/OL]. http://www.intel.com, 2007-12-31/2008-2-15

只读事务启动时，系统赋予该事务一个开始时间戳，只读事务通过读取小于该时间戳的元组的最近提交版本获得结果，由于采用多版本管理，只读事务不需要对数据进行加锁.更新事务对数据的更新在元组的新版本上进行，当更新事务开始进行数据更新的时候，在元组级上进行数据的拷贝（tuple level copy-on-update），该事务对该元组的后续更新，在已经拷贝的版本上进行.更新事务的串行性，通过两阶段锁协议实现.操作某个分区的更新事务必须首先获得该分区的加锁.

图2以实例说明上述事务调度方法的执行过程（Q:Query; U:Update Transaction).为了方便解释事务的串行执行，所有事务都存取同一个数据对象X.更新事务U1和U2同一个时刻开始执行，但是由于它们都存取同一个数据对象X，两阶段锁协议保证了它们的串行执行，即U1在T1时刻执行，在T2时刻提交，生成数据的新版本X1，U2得以在T2开始执行，在T3时刻提交，生成数据的新版本X2. Q1和U3在T3时刻开始执行，Q1获得T3时间戳，读取X的最近提交版本即X2返回结果，U3在X2的新版本上进行操作，在T4进行提交，生成新的提交版本X3. Q2、U4和U5在T4时刻开始执行，Q2获得时间戳T4，读取X3返回结果，U4和U5的串行性也通过加锁得到保证.值得注意的是，如果Q1是在U1提交以后U2提交之前开始，即Q1的开始时间戳大于T2而小于T3，那么事务的串行调度为U1Q1U2.由此可见，只读事务被串行调度到其启动之前已经提交的事务之后，和其启动后提交的所有事务之前.

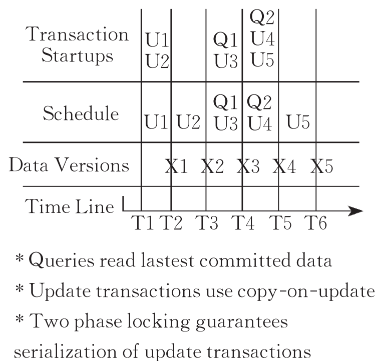



图2 事务的调度执行

采用上述并发控制策略，由于只读事务不需要进行数据加锁，不会和更新事务发生加锁冲突，提高了系统的并发度，从而提高系统吞吐量.

### 3.3 减少检查点操作的空间开销

在多版本管理的基础上实现一致检查点操作，内存的消耗成为算法设计的主要性能指标. PB-TCC-DV一致检查点技术方案，采用两级版本管理机制.每个数据分区共享一个版本缓存，在版本缓存里，除了元组的最近提交版本以外，老元组版本可以被操作该数据分区的更新事务重用，当有长查询事务需要某些元组版本，而版本缓存已经满的时候，才有必要把元组版本迁移到版本池.更新密集型应用的负载包含大量的短事务（查询的元组数量少），长查询事务数量较少（OLAP应用可以通过建立数据仓库予以支持），通过版本的快速重用，避免了版本的堆积，减少了内存空间的消耗.同时，PB-TCC-DV技术方案通过版本共享技术进一步减少版本管理的空间开销.

#### 3.3.1 两级版本管理

版本管理的基本粒度是元组，PB-TCC-DV采用两级管理机制进行版本管理，即数据分区（Data Partition）级的版本缓存（Version Cache）和数据段（Segment）级的版本池（Version Pool）（图1）.

①利用版本缓存，加快空间重用：版本缓存（version cache）的大小是一个数据页面，隶属于某个数据分区，对该分区的临时元组版本进行保存，目的是加快版本空间的及时回收利用.内存数据库的事务执行时间很短，存取同一个数据分区的事务，可以及时重用已经过时的版本缓存里面的旧版本.比如，更新事务U1修改数据元素X，并且提交，形成版本X1，事务U2接着修改数据元素X1，并且提交，形成版本X2.当其它更新事务（比如U3）需要版本池的空间进行元组的复制时，因为数据元素X的最新版本是X2，X1占用的空间可以被及时重用（图3）.

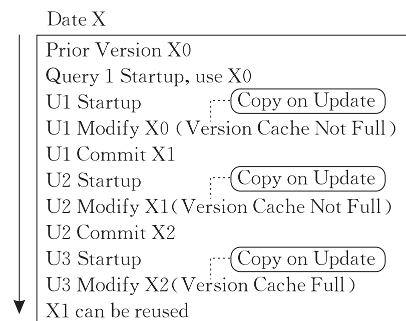



图3 版本重用

②利用版本池支持长查询事务：版本池（Version Pool）隶属于整个数据段，用于支持长查询事务.长查询事务存取更多的元组，执行的时间更长，需要把元组的旧版本保留起来，以支持这些事务的

执行.版本池采用先进先出的方式(FIFO)进行管理,当最早(oldest)的查询事务结束时,不再需要的元组版本由垃圾回收线程进行空间回收.

元组版本通过指针链接起来(图 4).当数据分区的版本缓存不能满足新事务的版本空间要求,旧元组版本被及时迁移到数据段的版本池里,保证长查询事务的正常运行.

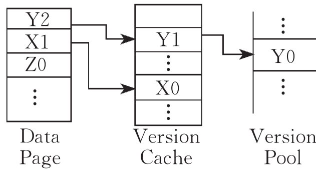



图 4　版本管理

采用两级版本管理机制,版本空间的回收利用得以加快,减轻长查询事务所导致的版本池上的版本堆积,从而减少检查点操作的空间开销.

3.3.2　动态版本共享

动态版本共享技术(dynamic version sharing)一般用来提高系统的并发度.在本文提出的方案中,我们利用动态版本共享技术,在多个查询事务(query)间共享某些元组版本,进一步减少版本管理的空间开销,使得一致检查点的空间代价进一步降低.

动态版本共享的工作原理是,通过把若干查询事务成组地进行执行(以一个启动时间戳),那么需要维护的数据库一致视图减少,需要保留的元组旧版本相应减少.如图 5 所示,查询 Q2 进入系统,时间戳为 T4,这个查询可以按 T4 时间戳运行,或者和查询 Q1 共享数据库的逻辑视图,使用其时间戳 T2.在 Q2 启动以后,一个更新事务 U2 对数据元素 X 进行了更新,生成新的版本 X2,并且在 T5 时刻提交.如果 Q2 共享 Q1 的时间戳,因为 X 的最新版本是 X2,那么 X 的旧版本 X1 就不需要进行保存,X1 所占用的空间就得到尽快的回收利用.

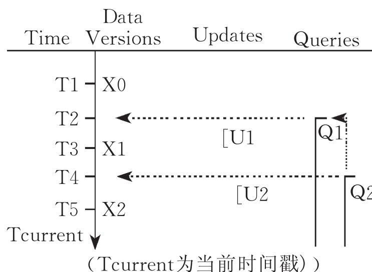



图 5　动态版本共享

## 3.4　无堵塞的检查点操作

PB-TCC-DV 检查点技术方案,以数据分区作为检查点操作的基本单位,检查点操作以一个只读事务执行,该事务启动的时候获得一个启动时间戳,读取数据分区该时间戳以前的提交数据,更新磁盘上过时的检查点文件.每个数据分区的检查点文件,是检查点事务启动时刻的事务一致的快照(transaction-consistent snapshot).检查点文件采用 Ping-Pong 方式进行维护,即在硬盘上维护分区的两个检查点文件,当新的检查点文件完整地写入磁盘以后,当前检查点文件指针做相应改变,而原有的检查点文件作为上一个检查点文件,以备重用.

更新事务执行过程中,相应数据分区的更新频率被记录下来.数据分区的检查点操作频率依据其更新频率进行计算.假设数据库包含 $N_{\text{part}}$ 个数据分区,数据分区 $i$ 的更新频率(update frequency)为 $UF(i)$,数据分区的检查点操作(checkpoint frequency)频率 $CF(i)$ 使用式(1)进行计算.

$$CF(i)=\left\lceil Npart\times\frac{UF(i)}{\sum_{i=1}^{Npart}UF(i)}\right\rceil \tag{1}$$

检查点操作算法的执行步骤简述如下.

**算法 1.**　一致检查点操作算法.

输入:各个数据分区的更新频率 $UF(i)$

输出:检查点文件

算法步骤:

1. 统计每个数据分区自上一个检查点操作以来的更新频率 $UF(i)$, $i=1,\cdots,N_{\text{part}}$.

2. 使用式(1)计算各个数据分区的检查点频率 $CF(i)$, $i=1,\cdots,N_{\text{part}}$.

3. 找出 $CF$ 最大的分区,该数据分区的检查点操作频率大于 0,比如分区 $i$,有 $CF(i)>CF(j)$, $j=1,2,\cdots,i-1,i+1,\cdots,N_{\text{part}}$,且 $CF(i)>0$.

4. 对该数据分区进行事务一致的检查点操作,一致检查点操作执行的过程是:

4.1. 在数据分区 $i$ 的日志中写入 BC(Begin Checkpoint)日志记录.

4.2. 读取数据分区 $i$ 各个数据页面的提交数据,首先保存在内存缓冲区中,当所有的页面读取完毕,把缓冲区刷新到外存中,形成新的检查点文件.

4.3. 在数据分区 $i$ 的日志中写入 EC(End Checkpoint)日志记录.

4.4. 在元信息中登记数据分区 $i$ 的本次检查点文件指针.

5. $CF(i)$减 1.

6. 检查是否还有某个数据分区的检查点操作频率大于 0,即存在 $CF(i)>0$,则转步 3,否则继续.

7. 把各个数据分区的检查点频率置为 0,然后转步 2.

上述算法保证,被频繁更新的数据分区,相应地被频繁地刷新到磁盘上,而较少更新的数据分区,其

检查点操作次数相应减少.由此可见,PB-TCC-DV检查点技术方案能够很好地处理存取倾斜(skew access pattern)状况.

在动态多版本并发控制机制下,检查点操作和更新事务使用不同的数据版本,解除了检查点操作和正常事务处理的紧密耦合(图6),于是检查点操作对正常事务处理的干扰大大减小,使得系统获得更高的吞吐量.

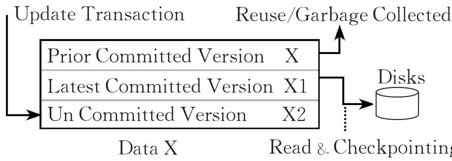



图6 解除检查点操作和正常事务处理的耦合

### 3.5 恢复过程中的系统可用性

在检查点操作的过程中,活动事务并发执行,当这些事务最后进行提交,其日志记录将到达稳定存储器,保证了系统的可恢复性,日志记录根据数据分区ID进行分类.恢复过程中,检查点文件装载进内存以后,只需要一趟日志文件扫描和处理(redo pass),把日志记录应用到数据分区的内存影像上,把上次检查点之后的提交事务重做一遍,即可把数据分区恢复到最近一致状态.

数据分区根据恢复优先级进行装载和恢复.具体实现策略是,恢复管理器根据当前事务对数据的请求和数据分区的更新频率,计算不同数据分区的恢复优先级(priority),启动一系列线程小组,负责排名前 $N_{\text{recovery}}$ 个数据分区的恢复,以期降低事务响应时间.假设内存数据库有 $N_{\text{part}}$ 个数据分区,记为 $P(j)$,各个数据分区的更新频率分别为 $UF(j)$,$j=1,2,\cdots,N_{\text{part}}$.当前系统正在运行的事务数量是 $M_{tx}$,记为 $T(i)$,各个事务在系统中的等待时间分别是 $WT(i)$, $i=1,2,\cdots,M_{\text{tx}}$. $A_{ij}$ $(i=1,2,\cdots,M_{\text{tx}};j=1,2,\cdots,N_{\text{part}})$为一个存取矩阵,如果 $T(i)$在等待进行 $P(j)$的存取,则 $a_{ij}=1$,否则为0.各个数据分区的恢复优先级计算公式为

$$Priority(j)=\alpha\frac{UF(j)}{\sum_{j=1}^{N_{\text{part}}}UF(j)}+(1-a)\sum_{i=1}^{M_{\text{tx}}}a_{ij}\frac{WT(i)}{\sum_{i=1}^{M_{\text{tx}}}WT(i)}\quad(2)$$

公式表示,更新频率越高,数据分区的恢复优先级越高,事务等待时间越长,等待的事务数量越多,数据分区的恢复优先级越高,公式中的 $\alpha$ 参数表示更新频率和事务等待时间的权重关系,一般取 $\alpha=0.3$.一旦有数据分区恢复完成,则把线程小组分配给余下的最高装载优先级的数据分区,恢复优先级必须重新计算,因为事务的等待时间发生了变化.各个数据分区的恢复并行执行,保证新事务的数据存取请求得到尽快的满足,降低其响应时间.

## 4 实 验

为了测试PB-TCC-DV检查点操作算法对系统的吞吐量和事务的响应时间的影响,我们利用J-SIM[①]仿真软件建立了内存数据库仿真系统,这是一个闭环队列网络仿真系统(closed queue network simulation system).

### 4.1 仿真系统模型与参数

仿真系统的构成如图7所示,包括事务处理、资源管理(图7,RM)、日志处理、检查点处理和恢复处理等关键模块.仿真系统的主要参数如表1所示.

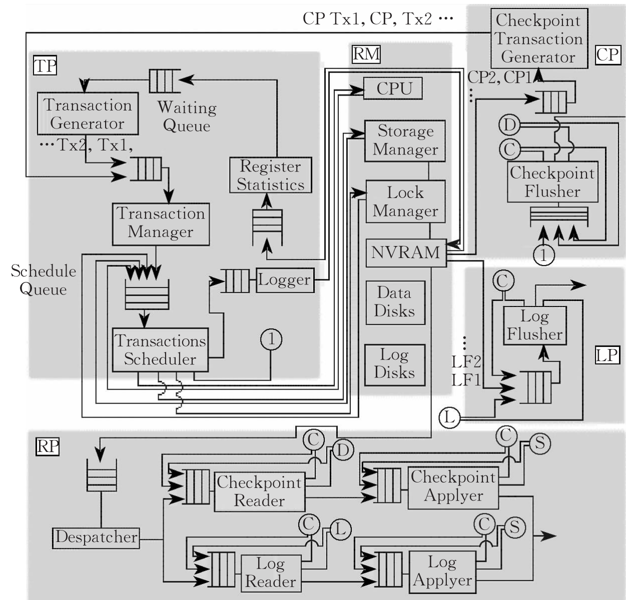



图7 仿真系统模型

事务处理流程(图7,TP):首先,事务生成器生成与MPL相同个数并发事务,交给事务管理器,由其进行调度.当有事务完成,事务生成器生成一个新的事务,以维持系统的并发事务数量.在动态多版本两阶段锁协议控制下,读事务不需要加锁,读取元组的提交版本,返回结果;更新事务更新数据的时候,

① J-SIM Developer Team. J-SIM User Manual [EB/OL]. http://www.j-sim.zcu.cz. 2006-12-31/2008-2-15

需要向锁管理器申请对数据分区进行加锁. 如果加锁的申请立即得到授权，则进行数据的具体操作，否则在堵塞队列里等待，其它事务完成以后唤醒. 更新事务对元组的更新在未提交的(uncommitted)元组版本上进行.

表 1　仿真系统参数设置

| 参数 | 含义 | 取值(可能值) |
|---|---|---|
| CPU# | CPU 的数量 | 2 |
| CPU Speed | CPU 的 MIPS | 20K MIPS |
| NVRAM access time | NVRAM 的存取时间 | 226μs(512B) |
| NVRAM Capacity | NVRAM 的容量 | Unlimited |
| Disk access latency | 磁盘的存取延迟 | 4ms |
| Disk seek latency | 磁盘的定位时间 | 8ms |
| Disk bandwidth | 磁盘的传输带宽 | 36MB/s |
| Data Disks# | 数据磁盘的数量 | 2(1,2,4,6,8) |
| Log Disks# | 日志磁盘的数量 | 2(1,2,4,6,8) |
| DB Size | 数据库的大小 | 131072pages(2GB) |
| Partition Size | 数据分区的大小 | 32pages |
| Page Size | 页面大小 | 16KB |
| Tuple-Size | 元组大小 | 256B |
| Version-Chasing | 顺着版本链搜索某个版本的指令开销 | 100 Instructions |
| Update Transaction Percentage | 更新事务占负载的比率 | R20U80 |
| Transaction Size | 事务存取的记录数量 | 5～10tuples |
| MPL | Multi Programming Level | 128 |
| Access Pattern | 存取模式 | 80:20 |
| Tuple Select Instructions | 存取元组的指令开销 | 450 Instructions |
| Tuple Update Instructions | 更新元组的指令开销 | 450 Instructions |
| Startup Instructions | 初始化操作的指令开销 | 1000 Instructions |
| Terminate Instructions | 结束操作的指令开销 | 2000 Instructions |
| Lock Instructions | Lock/Unlock 的指令开销 | 300 Instructions |
| Latch Instructions | Latch/Unlatch 的指令开销 | 30 Instructions |
| IO Instructions | 一个 IO 操作的指令开销 | 5000 Instructions |

备注：R20U80 表示只读事务(Read Only)占 20%，更新事务(Update)占 80%.

日志处理建模(图 7,LP)：在正常事务处理的过程中，事务所生成的日志由事务自身进行管理，以避免全局日志(global log tail)的竞争. 当事务提交的时候，日志记录首先记入非易失性内存，加快事务提交. 非易失性内存里的日志记录，按照所操作的数据分区 ID 进行分类，由后台进程异步地写入到多个日志磁盘，以实现日志的快速记录.

检查点操作建模(图 7,CP)：检查点操作以数据分区只读事务(read only transaction)的形式实现，统一由事务管理器进行处理. 在 CPU 资源的使用上，检查点事务一般由恢复 CPU(recovery CPU)进行服务. 该事务读取数据分区各个页面的提交数据，保存在内存缓冲区中，然后把 IO 请求放置到 data disk 的队列里等待其进行 IO 操作服务，完成检查点文件的刷新.

数据分区恢复(图 7,RP)：在恢复过程中，一旦关键的元数据装载完成，系统就可以接纳新的事务. 新事务需要存取的数据分区，如果还没有载入内存，事务在存储引擎的存取队列里堵塞等待，数据分区被载入并且恢复到一致状态以后，重新唤醒该事务的执行. 由于检查点操作以数据分区为单位进行，所以不同分区的检查点文件可以分布到不同的磁盘上；隶属于不同数据分区的 Redo 日志信息，也分布到不同的日志磁盘上，利用数据和日志的分布，实现基于数据分区的按需恢复.

负载建模：我们参考文献[9]，以电信运营应用为对象进行了负载建模. 电信业务应用的特点是，事务存取的数据量一般比较少，但是要求极快的响应时间. 在负载的构成上，包括更新事务和只读事务两类，通过调整 Update Transaction Percentage 来调整两类事务的比例关系. 只读事务读取一定数量的元组，提交结束；而更新事务则读取一定数量的元组，进行更新，然后进行提交. 只读事务和更新事务读取的元组数量通过 Transaction Size 参数进行控制，该参数在 5～10 之间符合均匀分布，这样的参数设置，捕抓了电信运营应用的基本特点. 在实验中，我们采用 R20U80 的事务混合比例，即只读事务占 20%更新事务占 80%，模拟更新密集的应用场景.

## 4.2　实验结果

### 4.2.1　正常事务处理的系统吞吐量

我们通过实验，比较了系统进行分区模糊检查点操作和进行分区一致检查点操作(本文方案)时的读写事务的吞吐量，以比较这两种检查点操作算法对正常事务处理的干扰，实验结果如图 8 和图 9 所示.

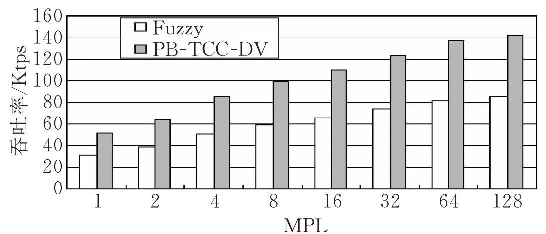



图 8　查询事务吞吐量(R20U80)

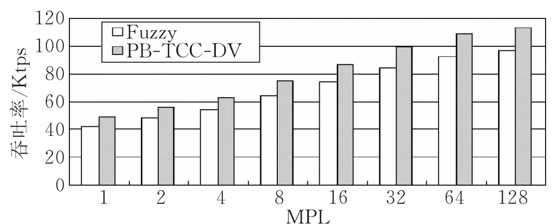



图 9　更新事务吞吐量(R20U80)

在只读事务方面，PB-TCC-DV 方案由于只读事务不需要进行加锁，获得了更高的性能，当 MPL 参数分别为 1、2、4、8、16、32、64、128 时，PB-TCC-DV 方案的吞吐量分别比 FUZZY 方案高 61%，67%，69%，67%，65%，68%，67%，63%. 在更新事务方面，由于避免了来自读事务的加锁冲突，PB-TCC-DV 方案的写事务吞吐量比 FUZZY 方案有 17%，13%，17%，15%，18%，18%，19%，17% 的提高.

由此可见，由于采用动态版本管理技术进行事务调度，避免了读写事务的加锁冲突，检查点事务和更新事务使用不同的元组版本，解除了检查点操作对正常事务处理的堵塞，提高了内存数据库系统的吞吐能力.

4.2.2 空间开销

PB-TCC-DV 检查点技术方案，在多版本管理的基础上实现一致检查点. 相对于模糊检查点技术，PB-TCC-DV 方案必须维护元组的旧版本，以支持只读事务和检查点操作. 空间开销的大小是否在可以接受的范围之内，是本文提出的技术方案是否可行的重要前提.

我们以不同的读写事务混合比例，进行检查点操作仿真，在仿真过程中，我们记录了版本缓存和版本池中临时版本的数量，并且计算临时版本占整个数据库数据量的比率，数据库的记录数量是 8M，结果如表 2 所示.

**表 2 不同更新事务比率的版本空间开销**

| 更新 Tx 比率/% | 空间开销(占整个数据量的比重)/% | 旧版本数 |
|---|---|---|
| 10 | 2.27 | 190589 |
| 20 | 3.25 | 272462 |
| 30 | 4.58 | 383863 |
| 40 | 5.49 | 460367 |
| 50 | 6.80 | 570425 |
| 60 | 8.11 | 680484 |
| 70 | 9.01 | 755646 |
| 80 | 10.19 | 854967 |
| 90 | 10.99 | 922076 |

从表 2 可以看出，当更新事务的比率增大时，由于更新事务需要在元组的新版本上进行操作，系统所需要维护的元组的版本数量不断增加. 当更新事务的比率为 80%的时候，用于维护元组版本的空间开销，占整个数据库数据量的比重为 10.19%. 这个空间开销是可以接受的，因为内存的容量越来越大，同时价格在不断地下降(具有 TB 级内存的系统已经成为现实)，使用一部分内存空间进行版本管理，以换取事务处理性能和检查点操作效率的提高是值得的.

PB-TCC-DV 技术方案的本质是以空间换时间，即通过利用一部分内存进行版本管理，以换取检查点操作和事务执行效率的提高. 试验证明，内存开销在可接受的范围之内，而系统性能的提高是可观的.

本文提出的技术方案，在版本管理的空间开销方面(R20U80 负载)与 SIREN 的空间开销相当，在系统吞吐量方面，比模糊检查点技术高 27%左右，比 SIREN 略低(30%)，但是本文提出的方案，通过数据分区的独立恢复，实现了恢复过程中的系统可用性，这是本文提出的方案和 SIREN 系统的主要区别.

4.2.3 恢复过程中的系统可用性

图 10 显示恢复过程中的系统吞吐量. 数据库的大小是 2GB，上次检查点以来的日志信息是 373MB，事务的存取符合 80∶20 规律，即 80%的事务存取 20%的数据.

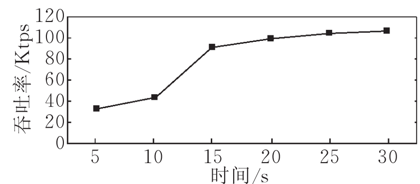
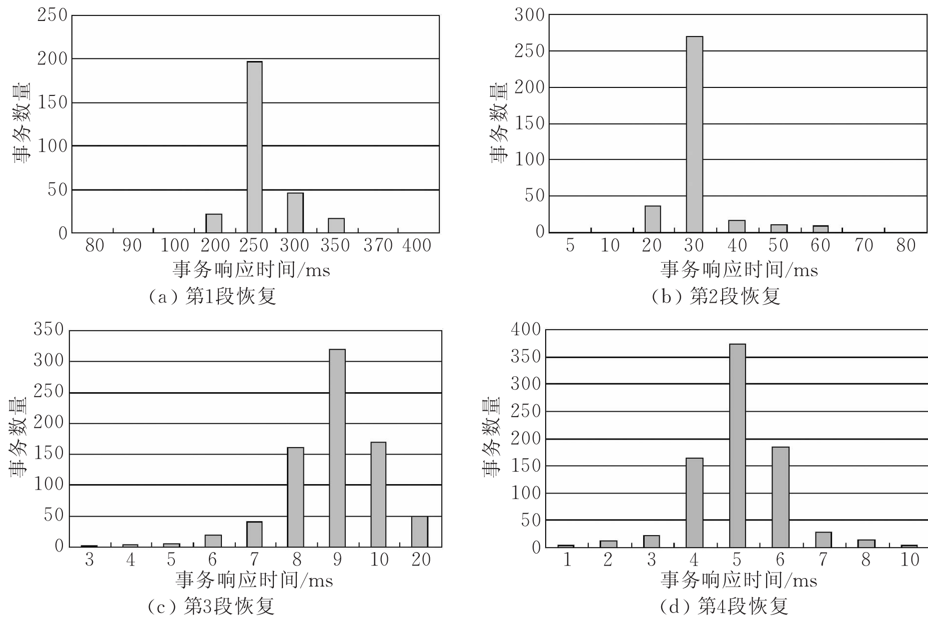



图 10 恢复过程中的系统吞吐量(R20U80)

系统在元数据恢复完毕之后，立即可以开始处理新事务. 在数据库恢复的比率较低的情况下，系统吞吐量受到了一定的影响. 在恢复过程中的前 5s、5～10s，系统的平均吞吐量分别为 33Ktps(transactions per second)和 43Ktps. 当数据库恢复到一定比率时，事务所存取的热点数据(hot spot data)已经全部驻留在内存中，系统的吞吐量迅速增长. 在恢复过程中的 10～15s、15～20s、20～25s 以及 25～30s，系统的平均吞吐量分别为 90Ktps、98Ktps、104Ktps、107Ktps，最后达到 117Ktps(图 10).

图 11 显示恢复过程中的事务响应时间分布. 我们把恢复过程划分成 4 个阶段，各个阶段的事务响应时间分布分别如图 11(a)～(d)所示. 在系统恢复初期，由于新事务的运行需要等待其所存取的数据分区装载到内存并且恢复到一致状态，所以响应时间有一定的延迟，平均事务响应时间是 276ms. 当事务处理所存取的热点数据不断被装载到内存并且恢复到

可用状态后，内存数据库事务处理的性能得到有效的发挥，响应时间锐减到 5.3ms. 在恢复过程的 4 个 1/4 时间段里，所执行的事务数量分别是 285K、343K、770K 以及 790K，事务的平均响应时间分别为 276ms、32ms、9ms、5.3ms. 当所有的数据都装载到内存里，内存数据库系统将全速运行，达到预期性能.



(a) 第1段恢复　(b) 第2段恢复　(c) 第3段恢复　(d) 第4段恢复

图 11　事务响应时间分布(R20U80)

# 5　总　结

本文提出针对内存数据库的事务一致分区检查点技术方案 PB-TCC-DV. PB-TCC-DV 方案在元组一级进行多版本管理，避免了检查点操作对正常事务处理的堵塞，同时利用两级版本管理机制以及动态版本共享技术，节省了版本管理的空间开销. 根据数据分区更新频率的不同，决定其检查点操作的频率，保证最近提交的数据不断地刷新到磁盘，使得恢复过程中需要处理的日志数量大大减少；同时，事务一致检查点使得 Redo Only 日志记录成为可能，恢复过程只需对日志进行一遍扫描处理，加快了恢复过程. 基于优先级的数据分区恢复，保证了新事务的数据存取请求得到优先的满足，提供了恢复过程中的系统可用性. 实验结果显示，PB-TCC-DV 在付出占数据库容量大约 11% 的空间开销的同时，获得了比传统的模糊检查点技术高 27% 的系统总吞吐量. 在内存容量越来越大，价格不断下降的情况下，付出这样的空间开销以获得事务处理和检查点操作的性能提升是值得的.

## 参　考　文　献


[1] Boncz P A, Kersten M L, Manegold S. Breaking the memory wall in MonetDB. Communications of the ACM, 2008, 51(12): 77-85

[2] Bober P M, Carey M J. Multiversion query locking//Yuan Le-Yan. Proceedings of the 18th International Conference on Very Large Data Bases. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992: 497-510

[3] Lin J L, Dunham M H. A performance study of dynamic segmented fuzzy checkpointing in memory resident databases. Department of Computer Science and Engineering, Southern Methodist University, Dallas, USA: Technical Report 96-CSE-14, 1996

[4] Jing Huang, Le Gruenwald. An update-frequency-valid-interval partition checkpoint technique for real-time main memory databases//Bestavros A, Lin K-J, S H S eds. Proceedings of the 1st International Workshop on Real-Time Databases: Issues and Applications. Newport Beach, California, USA: RTDB Endowment, 1996: 130-137

[5] Liao Guo-Xiong, Liu Yun-Sheng, Xiao Ying-Yuan. A partition fuzzy checkpointing strategy for real-time main memory databases. Journal of Computer Research and Development, 2006, 43(7): 1291-1296(in Chinese)
(廖国琼，刘云生，肖迎元. 实时内存数据库分区模糊检查点策略. 计算机研究与发展，2006，43(7)：1291-1296)

[6] Liedes A P, Wolski A. SIREN: A memory-conserving, snapshot-consistent checkpoint algorithm for in memory databases//Liu Ling, Reuter Andreas, Whang Kyu-Young, Zhang Jian-Jun eds. Proceedings of the 22nd International Conference on Data Engineering. Washington, DC, USA: IEEE Computer Society Press, 2006: 99-107

[7] Mohan C, Pirahesh H, Lorie R. Efficient and flexdible methods for transient versioning of records to avoid locking by read only transactions//Michael Stonebraker. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: ACM Press, 1992: 124-133

[8] Lehman T J, Gottemukkala V. The design and performance evaluation of a lock manager for a memory-resident database system (Chapter of Performance of concurrency control mechanisms in centralized database systems). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995: 406-428

[9] Blott S, Korth H F. An almost-serial protocol for transaction execution in main-memory database systems//Bressan S, Chaudhri A B, Lee Mong-Li, Xu Yu Jeffrey, Lacroix Z eds. Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong, China: VLDB Endowment, 2002: 706-717

**QIN Xiong-Pai**, born in 1971, Ph.D. candidate, lecturer. His research interests include Web information retrieval, database tuning, and high performance main memory database systems.

**XIAO Yan-Qin**, born in 1974, Ph. D. candidate. Her current research interests include main memory database, OLAP, and high performance databases.

**CAO Wei**, born in 1975, Ph. D. candidate, teaching assistant. Her research interests include database tuning, and self-namaging database systems.

**WANG Shan**, born in 1944, professor, Ph. D. supervisor. Her current research interests include high performance database and knowledge system, data warehousing technology, and grid data management. She has extensive publications in theses areas.

**Background**

In update intensive applications, main memory database systems produce large volume of log records, periodically checkpointing is necessary to reclaim log disk space, in the meantime, vulnerability of physical main memory makes checkpointing important for the recoverability of database systems.

Fuzzy checkpointing variants for main memory database systems have been deeply investigated since 1980s. Little could be done to improve them. The major disadvantage of fuzzy checkpointing is that two passes of log scanning are needed to restore the database, and new transactions cannot get executed until all log records are processed. Consistent checkpointing is regaining research effort in recent years mainly due to larger and cheaper memory, consistent checkpointing has several nice properties, it enables redo only logging which speedups system restart, partition based checkpointing supports partition recovery which is critical for system availability during recovery.

The capacity of memory gets larger, it is reasonable to use some memory in multi versioning for consistent checkpointing. A partition based transaction consistent checkpointing scheme (PB-TCC-DV) is proposed. A tuple-level dynamic multi-versioning concurrency control protocol is used to schedule transactions. Under the control of the protocol, checkpointing activities incurs little interfering with normal transaction processing, the system achieves high throughput. The two-level version management scheme and dynamic version sharing technique are used to further reduce space overhead. The experiment results show that PB-TCC-DV checkpointing scheme outperforms Fuzzy checkpointing by about 27% with 10.99% space overhead incurred. The scheme excels in terms of interfere with normal transaction processing, supporting fast restart, providing system availability during recovery, as well as handling skew access pattern.

The research is supported by the National Natural Science Foundation of China under Grant No. 60496325 and No. 60503038. Also it is partially supported by the international cooperation program with HP Lab China (The Project of Large Scale Data Management). The authors are exploring novel, effective, and efficient ways to mange large scale of data under both the rapid improvements of hardware and the more and more challenging demands of database applications.

The team has made much progresses and achievements in this research area. They have setup a main memory database test bed that is based on MonetDB as well as a simulation environment, and have published a number of papers which are involved in database performance behaviors on different hardware architecture (CPUs), query processing techniques for new hardware, domain benchmark for main memory database systems, parallel recovery of main memory database and so on.

The research in this paper is focusing on looking for a checkpointing scheme with high efficiency for update intensive main memory database systems. The experiment results show that the scheme is superior over fuzzy checkpointing scheme with acceptable space overhead.