软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(Suppl.(2)):320−327

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

基于非易失性内存的持久化嵌入式内存数据库*

沙行勉[1,2], 陈咸彰[1], 马殿龙[1], 诸葛晴凤[1,2]

[1](重庆大学 计算机学院,重庆 400044)

[2](信息物理社会可信服务计算教育部重点实验室(重庆大学),重庆 400044)

通讯作者: 诸葛晴凤, E-mail: qfzhuge@gmail.com

摘 要: 随着近年来嵌入式应用的复杂化和多样化,工业界和学术界提出采用内存数据库满足嵌入式系统对数据处理性能不断提升的要求.然而,现有的内存数据库需要在磁盘或闪存等外存上持久化存储真实的数据库备份,并且以 I/O 操作的方式将数据库的更新操作同步回外存,有极大的性能开销.此外,这类数据库即便直接部署在新型非易失性内存(non-volatile memory,简称 NVM)中,也因为缺乏内存中的持久化机制而不能脱离外存.针对现有内存数据库的不足,提出一套面向 NVM 的持久化内存数据库设计方案.该方案直接用数据库独立管理 NVM,持久化存储 NVM 的空间信息以及内存数据库的元数据.依据该方案,在典型的内存数据库 Redis 的基础上实现了可在 NVM 上持久化的内存数据库.实验结果表明,该方案与既有 Redis 的持久化方案 AOF 相比,数据库的启动速度可提高 2 400 倍,关闭速度可提高 5 倍,set 操作的速度可提高 58 倍,delete 操作的速度可提升 34 倍.

关键词: 内存数据库;非易失性内存;持久化;嵌入式数据库;性能优化

中文引用格式: 沙行勉,陈咸彰,马殿龙,诸葛晴凤.基于非易失性内存的持久化嵌入式内存数据库.软件学报,2016,27(Suppl.(2)):320−327. http://www.jos.org.cn/1000-9825/16046.htm

英文引用格式: Sha EHM, Chen XZ, Ma DL, Zhuge QF. Design of persistent embedded main memory databases on non-volatile memory. Ruan Jian Xue Bao/Journal of Software, 2016,27(Suppl.(2)):320−327 (in Chinese). http://www.jos.org.cn/1000-9825/16046.htm

Design of Persistent Embedded Main Memory Databases on Non-Volatile Memory

Edwin H.-M. SHA[1,2], CHEN Xian-Zhang[1], MA Dian-Long[2], ZHUGE Qing-Feng[1,2]

[1](College of Computer Science, Chongqing University, Chongqing 400044, China)

[2](Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education (Chongqing University), Chongqing 400044, China)

Abstract: With the increasing complication and variation of embedded applications in recent decades, both industry and academia have been proposing to use main memory databases to meet the ever-growing demand of high-performance data processing in embedded systems. Nevertheless, the existing main memory databases all rely on the secondary storage, such as magnetic disks and SSD, to maintain the real database persistently. Moreover, they have large overhead in synchronizing the data between memory and storage via slow I/O operations. Even though they are deployed in the emerging Non-Volatile Memory (NVM), the existing main memory databases are unable to get rid of disks for their temporary in-memory data structures cannot survive system reboot. In solving the persistency problem of main memory databases, this paper proposes to manage the NVM by the database itself that is independent from the memory management system. The information of NVM and the metadata of the database are persistently fixed on the NVM. The proposed ideas are implemented in an open-sourced main memory database, Redis. The experimental results show that compared with the existing

* 基金项目: 国家高技术研究发展计划(863)(2015AA015304); 国家自然科学基金(61472052); 重庆市科技攻关项目(cstc2014yykfB40007)

Foundation item: National High-Tech R&D Program of China (863) (2015AA015304); National Natural Science Foundation of China (61472052); Chongqing High-Tech Research Program (cstc2014yykfB40007)

收稿时间: 2016-09-20; 采用时间: 2016-11-17

persistency scheme of Redis AOF, the proposed approach improves the speeds of starting database, closing database, set operations, and delete operations by 2400 times, 5 times, 58 times, and 34 times, respectively.

Key words: main memory database; non-volatile memory; persistence; embedded database; performance optimization

随着物联网和信息物理融合系统的发展,大量嵌入式应用,如过程控制、数据采集和实时监控等,对数据存储和处理速度的要求不断提高.作为这类应用中的一个重要组成部分,甚至是基础设施,嵌入式数据库(embedded database,简称 EDB)[1−5]的性能对嵌入式系统的性能至关重要.

传统数据库的性能受到慢速 I/O 的约束,突破这种性能约束的一个重要方法是采用内存数据库[6−10],即尽量在内存中完成数据的存储和处理.然而,现有内存数据库并非真正的持久化数据库.它们将真正的数据库持久化地保存在外存中,仅仅在运行时用 I/O 操作把数据库从外存装入内存中.因此,现有内存数据库是一种临时的数据库.为了持久化保存最近更新的数据,现有内存数据库需要不断地向磁盘写镜像或日志,造成较大的性能开销.

近年来发展出的新型非易失性存储技术(non-volatile memory,简称 NVM)[11−13],如相变存储器(phase change memory),具有掉电不丢失数据、可按字节访问以及接近 DRAM 的访问速度等特点,被视为新一代的内存[11],甚至文件存储[14−16].同样地,NVM 也具有帮助实现高性能的嵌入式内存数据库[17,18]的潜力.然而,现有的内存数据库没有设计可持久化的内存数据结构,即便是配置在非易失性内存中,也不能保证数据的持久化和可用性.具体而言,其问题主要表现在两个方面.其一,现有内存数据库使用的物理内存由操作系统管理,系统关闭并重启后,数据库无法保证仍然获得系统重启前所使用的物理内存;其二,现有的内存数据库的内存部分依赖于外存上的真实数据库,只是在运行时在内存中创建临时数据结构,这些临时数据结构在系统关闭时会被操作系统摧毁.所以,即便是将现有的内存数据库部署在 NVM 作内存的系统中,也无法在系统重启后恢复 NVM 中的数据信息.

为此,针对现有内存数据库的不足,本文提出了面向 NVM 的嵌入式内存数据库持久化方案,主要思想是由内存数据库独立管理 NVM,设计独立于操作系统的 NVM 内存管理机制,以及针对 NVM 的可持久化内存数据库元数据.相应地,数据库基本操作的流程也做出修改,直接用虚拟地址读写数据,并确保数据的一致性和持久化.基于该方案,内存数据库可以实现完全内存化,不需要在磁盘或 SSD 上保存数据库的备份,以及基于慢速 I/O 的备份操作.

为了验证我们提出的方案,本文以典型的开源内存数据库 Redis 为例,修改 Redis 的源码,在其中实现了我们提出的持久化内存数据库的基本原型.随后,本文在真实数据库中进行对比测试.测试结果表明,与 Redis 现有的基于外存的持久化方案 RDB 和 AOF 相比,我们的方案在启动时间、关闭时间和基本数据操作中均取得了显著的性能提升.例如,使用该方案,数据库的启动速度比 RDB 和 AOF 方案分别提高了 3 859 倍和 2 402 倍.

本文的主要贡献如下:

(1) 提出了在 NVM 中可持久化的内存数据库的设计思路和方案,包括独立的 NVM 内存管理机制、可持久化的关于内存数据库的元数据结构和保证数据一致性的基本操作流程等.

(2) 以 Redis 为例,实现了一个可持久化内存数据库的基本原型.

(3) 大量实验结果表明,我们的方案能够显著提升系统性能.

本文第 1 节对嵌入式内存数据库的相关工作进行总结,并分析现有内存数据库不能很好利用 NVM 的原因.第 2 节提出在 NVM 上可持久化的内存数据库的设计思想,并基于开源内存数据库 Redis 实现持久化的内存数据库.第 3 节测试本文实现的持久化内存数据库的性能,并与现有内存数据库做对比分析.最后总结全文.

1 相关研究

内存数据库通常在系统运行时将整个数据库加载到内存中,从而提高数据库运行时的性能.典型的内存数据库有 Mcobject 公司的 eXtremeDB[8],Oracle 公司的 TimesTen[6]和开源数据库 Redis[7]等.这些内存数据库的设计实现都是面向 DRAM.由于 DRAM 具有掉电丢失数据的性质,目前的内存数据库只能将数据库暂存在内存中,而真实数据库只能持久化地保存在磁盘或 SSD 等外存中.因此,数据库在打开和关闭时,都要在内存和外存之间

传输大量数据,其慢速 I/O 操作会极大地降低性能.

此外,为了保证数据的持久性和一致性,现有内存数据库在运行过程中需要把修改的内存数据备份到磁盘上的真实数据库.以 Redis 为例,其提供两种持久化数据库的机制.一是快照处理(snapshotting),在 Redis 的实现中称为 RDB 模式.在此模式下,Redis 在指定时间间隔内生成数据库在内存中的数据集的快照,并写回外存中的真实数据库.然而,RDB 模式不能保证最近一个时间间隔内执行的更新操作的持久化和一致性.如果在时间间隔内系统掉电,则 Redis 就会丢失最近一个时间间隔内的全部数据库操作.为了提供更好的数据持久化和一致性保障,Redis 提供另一种称为追加文件(append-only file,简称 AOF)的机制.AOF 机制可以把每一次查询操作都以日志的形式持久化地记录在外存中.然而,AOF 会发起大量 I/O 操作,极大地降低了系统性能.

即便是用 NVM 替换 DRAM,简单地让现有内存数据库运行在 NVM 上,也不能舍弃外存上的真实数据库和 I/O 操作.原因有二,其一是目前的内存数据库占用的物理内存都由操作系统的内存管理系统分配和回收,系统重启后全部内存信息都会与上一次系统的状态不同.其二,现有内存数据库普遍使用临时数据结构管理内存中的数据,并不会记录整个数据库的索引,每次重新启动系统时都要重新创建内存部分的数据库.所以,当系统重启后,即便数据仍然存在于 NVM 中,现有内存数据库也无法恢复数据的组织结构和存放位置.

为此,本文将针对现有内存数据库的不足,以 Redis 为例,探讨基于 NVM 的高效嵌入式内存数据库,并实现一个基本的持久化内存数据库.

2 设计与实现

本文提出将内存数据库直接存储在 NVM 中,设计针对 NVM 的元数据结构实现数据库的持久化,并重新设计实现主要操作的流程.

2.1 NVM的物理空间管理

目前内存数据库所使用的内存都是 DRAM,其分配和回收都由操作系统的内存管理系统负责.系统重启后,内存管理系统会收回所有内存并建立新的空闲管理结构.因此,要将内存数据库持久化地存放在 NVM 中,首先要保证数据库占用的物理内存不被系统的内存管理系统回收或再分配.本文提出单独划分 NVM 内存区域,并用数据库独立管理 NVM 的分配和回收.

为了持久化 NVM 空间的使用信息,我们在 NVM 设备的起始位置划分出一段区域,用来记录 NVM 存储空间的管理结构.每次系统重启后,就可以从 NVM 固定的起始位置恢复 NVM 的整体信息.内存数据库自身也需做类似处理,第 3.2 节将详细加以介绍.以 Redis[7]为例,我们的 NVM 管理方案如图 1 所示.在 Redis 内存数据库原架构的基础上,新增了一组针对 NVM 的管理(NVM metadata).

(1) 空闲链表(FreeList):用于索引 NVM 设备中的所有空闲物理页面.

(2) 细粒度内存分配器(NVMslab):用于分配、回收小于一个页面大小的 NVM 物理空间.

(3) 页面掩码(PageMask):用于记录每个页面当前的空间使用情况.

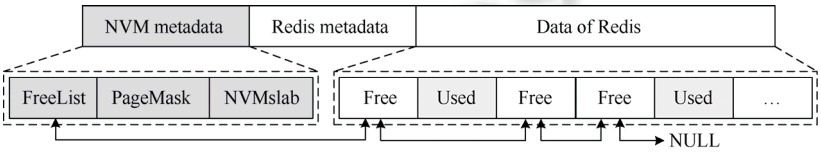


Fig.1 Metadata structure for managing the NVM of Redis

图 1 管理 NVM 的 Redis 元数据结构

为了便于管理,NVM 的空闲物理内存被划分为多个页面,单个页面的大小为 4KB.空闲页面用空闲链表连接起来.此外,作为基于一个 key-value 型数据库,Redis 中的一个对象往往仅需小于 4KB 的内存空间.为了提高 NVM 的空间利用率,本文提出使用细粒度内存分配器和页面掩码辅助管理 NVM 的物理空间.细粒度内存分配器

NVMslab 将一个页面拆分为 128 个用于分配回收的基本单元,一个基本单元的大小为 32 个字节.每当要分配一段小于 4KB 的空间时,首先判断分配器里面的空闲空间能否满足要求.如果不能满足要求,就从空闲页面链表中获取一个 4KB 的物理页给 NVMslab,然后再进行分配.页面掩码的结构类似于数组,每个元素对应一个 NVM 物理页,存放该页面的空间使用情况.在当前实现的版本中,每个元素是一个位图.一个 4KB 大小的空闲页对应在页面掩码中的位图大小为 128 位(bit).每从一个页面分配 32 字节,页面掩码中对应的位就置为 0;每从某个页面释放 32 字节,页面掩码中对应的位就置为 1.

当数据库收到分配 NVM 空闲空间的请求时,首先判断请求的大小.当请求的空间大于 4KB 时,以 4KB 为单位向上取整,从空闲链表中直接分配连续的空闲块.如果要分配的空间小于 4KB,就用细粒度内存分配器 NVMslab 配合页面掩码获取可用内存.NVMslab 尽量分配连续的物理内存给单个 key-value 存储对象,以提高对快表(translation lookaside buffer,简称 TLB)和 Cache 等硬件的利用率.例如,一个存储对象占用 256B 物理内存,如果 NVMslab 从 8 个不同的物理页面中分配 8 个 32B 的基本单元给该对象,访问该对象就需要查找 8 个不同的页表项.反之,如果使用同一个物理页面的基本单元,就只需在访问第 1 个基本单元时查找 1 个页表项.由于该页表项已经装载到 TLB 中,访问其后 7 个基本单元就可以直接利用 TLB 中的页表项高速访问物理内存.

在空闲链表中,用于链接空闲页面的信息都存在“空闲”空间中,所以不产生空间开销.而页面掩码需要存储在专有区域中,其空间开销约为 0.39%=128bit/4KB.

2.2 数据库信息的持久化

除了持久化 NVM 物理空间的信息之外,还需要持久化保存内存数据库的当前状态,其关键在于持久化地记录数据库在内存中的索引信息.

首先要确定的是需要持久化的信息.以 Redis 为例,Redis 可以维护多个子数据库,每个子数据库用一套数据结构组织起来,如图 2 所示.图 2 中仅保留各数据结构直接的连接关系,忽略其他描述属性的细节.

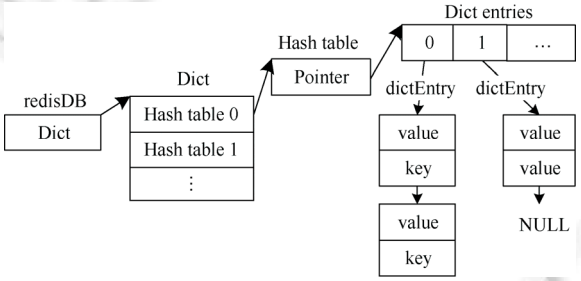


Fig.2 Main in-memory data structures of Redis

图 2 Redis 在内存中的主要数据结构关系

一个 Redis 数据库的最高层结构为 redisDB,其中存放了指向该数据库的哈希词典(Dict).由哈希词典逐层往下可以找到哈希表(Hash table),词典项(Dict entry),直到存放具体数据的 key-value 键值对.在 Redis 中,每个子数据库都独占一份哈希词典,指向多个 Redis 子数据库的哈希词典的结构体以数组的形式组织在一起,存放在 redisDB 中.由此可见,如果在每次系统启动时都能找到 redisDB,就可以进一步检索到整个数据库.所以,Redis 中需要持久化的信息就是 redisDB.

为此,在 NVM 紧随 NVM 元数据的位置留出一段固定的物理空间,以存储结构体 redisDB,如图 3 所示.该 Redis 里所有的哈希词典都用哈希词典数组(Dict array)组织起来,持久化地存放在 NVM 中.Redis 每次启动时,都固定使用这段空间的首地址找到 redisDB,然后通过 redisDB 定位到哈希词典数组,进而索引整个内存数据库.

其他内存数据库与 Redis 类似,都设有关于整个数据库的元数据信息或索引结构.当数据库独立管理 NVM 时,只要将这些元数据或索引结构存放在 NVM 的固定位置,就能在重启后恢复整个内存数据库.

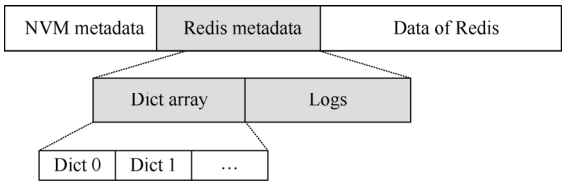


Fig.3 Metadata structures of persistent Redis

图 3 持久化的 Redis 元数据结构

2.3 修改数据库的基本操作

当 Redis 持久化地存放在 NVM 中时,对内存数据库的操作不需要再经过 DRAM.对数据的任何修改操作都直接持久化地保存到数据库中,不需要再同步回磁盘中,省去全部的慢速 I/O 操作.

由于所有操作直接作用在数据库中,需要重新考虑一致性问题.本文采用写日志(logging)的方式保证数据库的一致性.如图 3 所示,我们在 Redis 的元数据部分的哈希词典数组之后,增加了一块日志区域(log).每一个修改数据库的操作,诸如 Set 和 Delete,都以事务的方式记录在日志区域.

由于日志区域固定存放在 NVM 上,系统崩溃重启后就可以直接定位到日志区域,依次扫描日志并恢复数据库.与 Redis 现有的 RDB 方式不同,本文提出的方案在日志中记录了每个数据库操作的信息.因此,系统恢复时可以恢复到数据的最后一个一致性状态,这提供了更好的数据一致性保证.

此外,基于该方案,本文在 Linux 系统中实现了基于 Redis 的持久化内存数据库.首先,NVM 被设置为一块由数据库单独管理的内存设备.其次,我们为 NVM 设计实现了一套独立于操作系统的内存管理系统的空间分配、回收函数:*nvm_malloc*()和 *nvm_free*().本文实现的持久化内存数据库不再使用 Linux 系统的内存分配、回收函数,如 *malloc*()和 *free*(),而是使用 NVM 的空间分配和回收函数,从而避免数据库的真正数据被系统所破坏.

3 实验结果与分析

我们首先介绍实验设置,其次测试分析我们的方案的性能.

3.1 实验设置

本文在 Redis 开源代码的基础上实现了我们提出的持久化方案,包括增加持久化元数据新型、面向 NVM 的内存管理单元,并修改相关数据库操作等.对比方案是 Redis 既有的两种持久化机制,即 AOF 机制(表示为 Redis-AOF)和 RDB 机制(表示为 Redis-RDB).

实验平台的硬件环境是 3.30GHz Intel® i3 4 核处理器,内存为 32GB DDR3 DRAM,外存是 7 200rpm SATA 3.0 接口的磁盘.系统为 Ubuntu 14.04.实验用 DRAM 模拟 NVM,其中设置系统内存的大小为 8GB,NVM 内存区的大小为 24GB.

本文选取基本的 Redis 数据库操作对比各个方案的性能,包括数据库的启动时间、关闭时间、Set 操作时间和 Delete 操作时间.在实验中,为了测试启动时间和关闭时间,设置不同规模大小的数据库.在测试 Set 和 Delete 操作时,分别提交不同数量的相关操作.为了记录各种操作在系统中的真实性能,我们在各个版本的数据库源码中添加了时间戳.

3.2 实验结果与分析

3.2.1 数据库的启动时间

启动时间的实验结果如图 4 所示.实验结果表明,我们的方案(表示为 Ours)在所有情况下都优于既有 Redis.当数据库中有 107 条记录时,Redis 在 RDB 和 AOF 模式下的启动时间分别是我们的方案的 3 859 倍和 2 402 倍.

现有 Redis 在启动时有两种建立内存数据库的策略.其中,RDB 方式是通过读取磁盘上的镜像文件恢复(二进制文件)内存数据库,而 AOF 方式不但要读取镜像,而且要扫描 Log 文件才能恢复内存数据库.而我们的方案只需在 NVM 中固定位置读取数据库的元数据、扫描日志就可以恢复数据,不需要把数据库的镜像加载到 DRAM.

因此,我们的方案启动时间稳定在 4ms 以内,而 Redis 的既有方案启动时间会随着数据库规模的增大而增加.

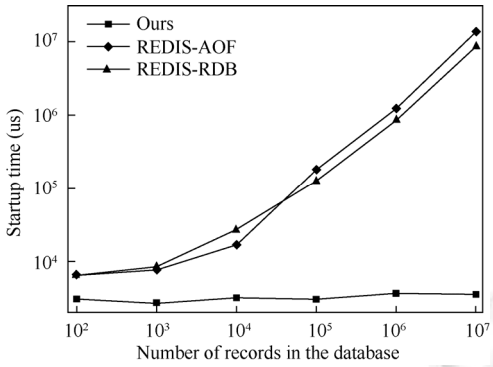


Fig.4 Comparing the startup time of Redis

图 4 比较 Redis 的启动时间

3.2.2 数据库的关闭时间

3 种配置下的 Redis 数据库的关闭时间如图 5 所示.与启动时间类似,在使用我们的方案时,Redis 的数据库关闭速度快于其他两种模式.其中,在 RDB 模式下,Redis 的关闭时间随着数据库规模的增大而增加,当数据库的规模增长到 107 个对象时,甚至比我们的方案慢 15 万倍.这是因为 RDB 模式要在关闭时把内存数据库的镜像写回外存,造成大量的 I/O 开销.

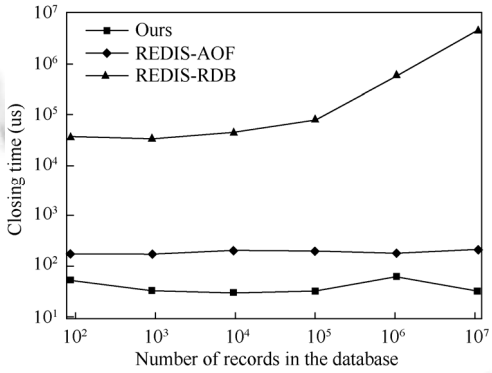


Fig.5 Comparing the closing time of Redis

图 5 比较 Redis 的关闭时间

我们的方案和 REDIS-AOF 的关闭时间不会随数据库规模的增大而增加,是因为这两种方案都通过写日志的方式将更新操作持久化地记录在数据库中,不需要在关闭数据库时同步数据.两者相比,AOF 比我们的方案耗时平均要多 5.2 倍,这是因为 AOF 需要释放内存数据库临时占用的 DRAM,而我们的方案不占用 DRAM 作为临时存储,只需要结束 Redis 进程即可,操作更少,开销更低.

3.2.3 Set 操作的性能

在 Redis 中,Set 操作用于在数据库中新增一个对象.由于对数据库的物理空间和组织结构都造成改变,所以需要保证操作的持久化和一致性.本文提出的方案和现有的 AOF 方案都可以保证任一个 Set 操作的一致性.REDIS-AOF 是把每个操作都同步写回外存,要经过慢速 I/O 操作.而在我们的方案中,由于数据库以及持久化在 NVM 中,所以省去全部 I/O 操作,仅用虚拟地址就可完成整个操作.如图 6 所示,相比之下,我们的方案性能比 REDIS-AOF 平均要高 58.6 倍.

尽管 REDIS-RDB 在此次测试过程中并未向外存写镜像,不对提交的 Set 操作提供一致性保证,我们的方案性能仍比 REDIS-RDB 的性能平均要高 47.6%.在 Set 操作的数量较少时,两者的性能差异并不大.当 Set 操作的

数量达到 107 时,我们的方案速度达到 REDIS-RDB 的 3 倍.这是因为当 Set 数量增大后,REDIS-RDB 向系统内存管理提交的分配请求大量增加,与系统中其他进程形成竞争,分配内存的开销增大.而在我们的方案中,由 REDIS 独占并管理 NVM,不存在与其他进程之间的竞争,分配内存的时间稳定,效率更高.

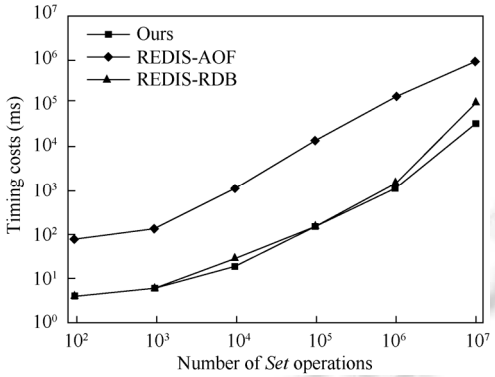


Fig.6 Comparing the timing cost of Set operation

图 6 对比 Set 操作的时间

3.2.4 Delete 操作的性能

Delete 操作是从数据库中删除对象,重新设置相关数据结构,涉及到数据库存储空间的回收和上层数据索引的修改.因此,Delete 操作也需要保证更新操作的持久化和数据的一致性.实验结果如图 7 所示.我们的方案执行效率比 REDIS-AOF 平均快 34 倍,同时又能达到和 AOF 一样的持久化和一致性保证.仍然是因为我们的方案完全是高速的内存操作,开销远低于传统方式所依赖的慢速 I/O 操作.与 REDIS-RDB 相比,我们的方案性能与之相当.然而,与 Set 操作相同,REDIS-RDB 仍然没有真实修改外存上的持久化数据库,不能保证任何删除操作的一致性和持久化.

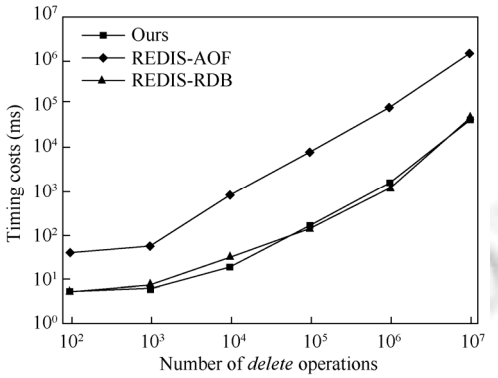


Fig.7 Comparing the timing cost of Delete operation

图 7 对比 Delete 操作的时间

4 总 结

基于非易失性内存的持久化嵌入式内存数据库具有高度提升嵌入式系统性能、降低系统能耗的潜力.然而,目前的内存数据库并没有考虑非易失性内存,不能真正实现在内存中的持久化和一致性保障.为此,本文深入探讨此问题,提出用内存数据库自主管理非易失性内存的思路,提出设计重启后可恢复内存数据库索引的可持久化元数据结构.此外,本文以经典的内存数据库 Redis 为例,设计并实现了一套针对 Redis 的可持久化的元数据结构、非易失性内存管理机制和提供高度一致性保证的操作方法.通过大量实验测试表明,我们的方案可以利用非易失性内存的特点,显著提升内存数据库的性能.事实上,本文提出的持久化内存数据库的方法还可以拓展到其

他需要持久化存储在新型非易失性内存的应用中.

References:

[1] Nori A. Mobile and embedded databases. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. 2007. 1175−1177. [doi: 10.1145/1247480.1247648]

[2] Liao GQ, Liu YS. Recovery schemes based on real-time logs for embedded real-time databases. Chinese Journal of Computers, 2007,30(4):672−679 (in Chinese with English abstract).

[3] Kang W, Sang HS. Power- and time-aware buffer cache management for real-time embedded databases. Journal of Systems Architecture, 2012,58(6-7):233−246. [doi: 10.1016/j.sysarc.2012.03.003]

[4] Kim GJ, Baek SC, Lee HS, Lee HD, Loe MJ. LGeDBMS: A small dbms for embedded system with flash memory. In: Proc. of the Int'l Conf. on Very Large Data Bases. 2006. 1255−1258.

[5] Liao RJ, Suei PL, Lu YF, Kuo TW, Lee CS. A signature-based grid index design for RFID main-memory databases. In: Proc. of the IEEE/IPIP Int'l Conf. on Embedded and Ubiquitous Computing. 2008. 519−525. [doi: http://doi.ieeecomputersociety.org/10.1109/EUC.2008.105]

[6] Oracle TimesTen. 2016. http://www.oracle.com/technetwork/products/timesten/overview/index.html

[7] Redis. 2016. http://redis.io/documentation

[8] EXtremeDB. 2016. http://www.mcobject.com/standard edition.shtml

[9] Abuelkheir M, Hayajneh M, Ali NA. Data management for the Internet of things: Design primitives and solutions. Sensors, 2013, 13(11):15582−15612.

[10] Luo L, Liu Y, Qian DP. Survey on in-memory computing technology. Ruan Jian Xue Bao/Journal of Software, 2016,27(8):2147−2167 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5103.htm [doi: 10.13328/j.cnki.jos.005103]

[11] Lee BC, Zhou P, Yang J, Zhang Y, Zhao B, Ipek E, Mutlu O, Burger D. Phase-Change technology and the future of main memory. IEEE Micro, 2010,30(1):131−141. [doi: 10.1109/MM.2010.24]

[12] Jung M, Shalf J, Kandemir M. Design of a large-scale storage-class rram system. In: Proc. of the ACM Int'l Conf. on Supercomputing. 2013. 103−114. [doi: 10.1145/2464996.2465004]

[13] Chen XZ, Sha EHM, Zhuge QF, Xue CJ, Jiang WW, Wang YG. Efficient data placement for improving data access performance on domain-wall memory. IEEE Trans. on Very Large Scale Integration Systems, 2016,24(10):3094−3104. [doi: 10.1109/TVLSI.2016.2537400]

[14] Condit J, Nightingale EB, Frost C, Ipek E, Benjamin L, Burger D, Coetzee D. Better I/O through byte-addressable, persistent memory. In: Proc. of the ACM Symp. on Operating Systems Principles. 2009. 133−146.

[15] Ou J, Shu JW, Lu YY. A high performance file system for non-volatile main memory. In: Proc. of the European Conf. on Computer Systems. 2016. [doi: 10.1145/2901318.2901324]

[16] Sha E, Chen XZ, Zhuge QF, Shi L, Jiang WW. A new design of in-memory file system based on file virtual address framework. IEEE Trans. on Computers, 2016,65(10):2959−2972. [doi: 10.1109/TC.2016.2516019]

[17] Suei PL, Lu YF, Liao RJ, Lo SW. A signature-based grid index design for main-memory rfid database applications. The Journal of Systems and Software, 2012,85(5):1205−1212. [doi: 10.1016/j.jss.2012.01.026]

[18] Ray S, Blanco R, Goel AK. Supporting location-based services in a main-memory database. In: Proc. of the IEEE Int'l Conf. on Mobile Data Management. 2014. 3−12. [doi: 10.1109/MDM.2014.7]

附中文参考文献:

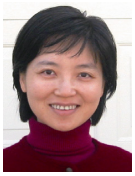
[2] 廖国琼,刘云生.基于实时日志的嵌入式实时数据库恢复策略.计算机学报,

[10] 罗乐,刘轶,钱德沛.内存计算技术研究综述.软件学报,2016,27(8):2147−2167. http://www.jos.org.cn/1000-9825/5103.htm [doi: 10.13328/j.cnki.jos.005103]

沙行勉(1964－),男,博士,教授,博士生导师,CCF高级会员,主要研究领域为大数据系统,并行计算和系统,嵌入式系统和软件.

马殿龙(1990－),男,硕士生,主要研究领域为内存数据库,非易失性内存.

陈咸彰(1989－),男,博士生,主要研究领域为嵌入式系统和软件,新型存储,文件系统.



诸葛晴凤(1970－),女,博士,教授,博士生导师,CCF专业会员,主要研究领域为大数据处理系统,嵌入式系统,物联网系统,优化算法.