计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200900071

NVRC:一种面向 NVM 的写限制日志方案

范鹏浩[1]　黄国锐[2]　金培权[1]
1 中国科学技术大学计算机科学与技术学院　合肥 230001
2 中国人民解放军 31002 部队　北京 100081
(fph327@mail.ustc.edu.cn)

摘　要　非易失性内存(Non-Volatile Memory,NVM)具有支持按字节寻址、持久性、存储密度高、读写延迟低等特点,因此成为解决 DRAM(Dynamic Random Access Memory)容量有限问题的首选技术。随着数据库系统中 NVM 的引入,传统的日志技术需要考虑如何适应 NVM 特性。首先总结了已有的面向 NVM 的日志技术研究,进而提出了一种尽可能限制 NVM 写操作的数据库日志方案 NVRC(Non-Volatile Record-updating with Cacheline)。文中提出了结合异地更新和原地更新的日志管理方案。具体而言,NVRC 在异地更新的"影子记录"的基础上,引入了"缓存行原地更新"策略,并通过代价分析选择合理的日志更新策略,从而减少对 NVM 的写操作。采用 DRAM 模拟 NVM 的方式在 YCSB 测试负载上进行了实验,并对比了 NVRC 与传统的 WAL(Write Ahead Log)以及 NVM 感知的 PCMLx(PCMLoggingx)方法。结果表明,NVRC 的 NVM 写次数在修改均匀的情况下比 WAL 和 PCMLx 分别减少了 54%和 17%,同时更新性能分别提升了 59%和 10%。

关键词:非易失性内存;数据库日志;原地更新;异地更新;影子记录
中图法分类号　TP311

NVRC:Write-limited Logging for Non-volatile Memory

FAN Peng-hao[1],HUANG Guo-rui[2] and JIN Pei-quan[1]
1 School of Computer Science and Technology,University of Science and Technology of China,Hefei 230001,China
2 PLA 31002,Beijing 100081,China

Abstract　Non-volatile memory (NVM) has the characteristics of byte addressing,persistence,high storage density,low read-write delay,etc.,so it becomes the preferred technology to solve the problem of limited DRAM(Dynamic Random Access Memory) capacity. With the introduction of NVM in database systems,traditional log technologies need to consider how to adapt to the characteristics of NVM. This paper first summarizes the existing research on NVM oriented log technologies,and then proposes a database log scheme called NVRC(Non-Volatile Record-updating with Cacheline) that limits NVM write operations as much as possible. This paper puts forward a log management scheme which combines out-place and in-place update. Specifically,on the basis of out-place-update-based shadow records,NVRC introduces the strategy of "in-place cache line update",and dynamically selects the log update strategy through cost analysis,so as to reduce the writes to NVM. This paper uses DRAM to simulate NVM to experiment on the YCSB benchmark,and compares NVRC with the traditional WAL(Write Ahead Log) and the NVM-oriented logging scheme PCMLx(PCMLoggingx). The results show that the number of NVM writes of NVRC is 54% and 17% less than that of WAL and PCMLx respectively,and the update performance improves by 59% and 10% respectively.

Keywords　Non-volatile memory,Database log,In-place update,Out-place update,Shadow record

1　引言

近年来,非易失性内存(NVM)作为一种新兴存储器得到了广泛的关注。NVM 并不是一种特定的存储器,而是代表一类具有共同性质的存储技术,包括 PCM,RRAM,STT-RAM 等。其中,目前发展较为成熟的是相变存储器(Phase Change Memory,PCM)[1-3]。NVM 通常具有如下特征:1)支持按字节寻址,因此可以像 DRAM 一样被 CPU 直接存取;2)读写速度快,远高于磁盘和闪存的读写速度;3)相比 DRAM 来说,具有持久性,掉电时也可以保存数据;4)非对称的读写延迟,写延迟通常高于读延迟;5)存储密度远高于 DRAM,因此在相同的尺寸下可以提供比 DRAM 更大的容量。

表 1 列出了 DRAM,NVM(以 PCM 为例)和 HDD 之间

到稿日期:2020-08-09　返修日期:2020-09-30
基金项目:国家自然科学基金(61672479,62072419)
This work was supported by the National Natural Science Foundation of China(61672479,62072419).
通信作者:金培权(jpq@ustc.edu.cn)

的特性对比。由于 NVM 目前尚处于快速发展阶段，因此表 1 中的数据也是逐年变化的，但总的特性没有较大的改变。

表 1 不同存储的对比[3-4]

Table 1 Comparison of different storages

Storage	DRAM	NVM(PCM)	HDD
durability	N	Y	Y
byte addressing	Y	Y	N
Read delay	25 ns	50～70 ns	3 ms
Write delay	30 ns	150～220 ns	3 ms
writes per bit	10^{18}	10^{8}～10^{12}	∞

NVM 由于具备了大容量、非易失、可以被 CPU 直接存取等特点，因此一旦被大规模装备在计算机系统中，将对现有的数据管理技术带来极大的挑战[5]。例如，在大容量 NVM 的支持下（目前 Intel 的傲腾持久性内存可以提供单机 8TB 的存储容量），我们可以构建不需要 HDD 支持的数据库系统，从而抛弃传统的以 I/O 作为主要代价模型的数据库算法。与此同时，NVM 也存在着写延迟高、读写不对称、写次数有限等缺点。如何在基于 NVM 的数据库系统中尽量发挥 NVM 的优势同时回避其缺点，是当前的一个研究重点。

本文主要研究面向 NVM 的数据库系统中的一个关键问题，即日志管理。数据库日志技术是保证数据库可恢复性和一致性的关键技术，但它在数据库系统的运行过程中开销较大。本文设计了一种新的日志管理方案 NVRC，利用 NVM 相比磁盘的字节寻址性优点来减少对 NVM 的写操作，从而在不影响数据库性能的前提下延长 NVM 的寿命。

总体而言，本文的主要工作和贡献包括：

1）提出了一种优化的面向 NVM 的数据库日志方案 NVRC，在使用 NVM 作为持久性存储的情况下，最小化 NVM 的写操作数。

2）对 NVRC 进行模拟实验，结果表明 NVRC 能够有效地减少 NVM 写操作。同时，本文提出了一些优化 NVRC 性能的策略。

本文第 2 节介绍了相关工作；第 3 节讨论了 NVRC 的详细设计和实现；第 4 节给出了实验结果；最后总结全文并展望了未来工作。

2 相关工作

随着 NVM 技术的日益成熟，国内外研究者在基于 NVM 的数据库技术尤其是日志技术上进行了很多研究，并提出了很多不同的结构和方案[6]。

数据库的事务恢复需要在事务完全提交之前保留事务开始前的数据，于是出现了先写日志[7]和影子分页[8]。先写日志技术是一种原地更新设计，原有的旧数据被新数据覆盖，为了恢复事务，需要通过记录日志来记录更新的操作，在恢复过程中通过读取记录在磁盘上的日志进行 Redo/Undo 操作，从而恢复数据以维护数据库的一致性。影子分页技术是一种异地更新设计，每次更新将记录写到磁盘的一个新的页里，从而保证旧数据不被覆盖，继续保存在磁盘中。如果事务中途中断并需要恢复，则将旧数据恢复使用，从而使数据库恢复到事务之前的一致性状态。

由于 NVM 与磁盘内存相比具有不同的特性，基于 NVM 的数据库日志设计面临着新的挑战。与磁盘相比，NVM 具有写寿命有限的缺点，同时具有按字节寻址的特点。由于写寿命有限，需要减少 NVM 的写操作以及实现 NVM 上不同位置的均匀写。而传统的磁盘数据库 WAL 策略一方面没有考虑均匀写的要求，另一方面其日志设计一般含有较多的数据冗余，不符合对 NVM 的要求。而影子分页技术以页为单位，忽略了 NVM 的字节寻址性，因此传统的影子分页技术也不适合基于 NVM 的数据库。按字节寻址的特点扩展了 NVM 的潜力，需要新的设计才能利用这一特点，从而挖掘出数据库的更多潜力。

目前，提出的针对 NVM 的日志改进设计总体上可总结为几种方式，包括简单替代、基于原地更新的策略、基于异地更新的策略等。

2.1 简单替代方案

简单替代方案的思想就是利用 NVM 的非易失性和比磁盘快得多的读写速度，用 NVM 替代原有的磁盘甚至 DRAM 来存储日志，从而大大提升读写速度，并且通过设计相关的算法和结构来保证数据的一致性。Fang 等[9]和 Kim 等[10]最早使用了该方法。Fang 等用 NVM 中的单个日志空间替代传统的内存日志缓冲区和基于磁盘的日志文件。使用 NVM，日志记录可以直接写入持久内存中，从而简化了传统的两层日志记录设计，提供了更好的并发性支持，并提高了事务延迟。他们还设计了新的数据结构和恢复算法以适应新的存储结构。Kim 等则是在已有的 IPL 算法[11]中引入 NVM，利用 NVM 优于闪存的读写速度和字节寻址性来获得更好的性能。

2.2 基于原地更新的日志方案

Pelley 等[12]不仅将 NVM 用于日志，还将 NVM 用于数据库数据以替代 DRAM，即在整个数据库的基础上将传统的 DRAM 和磁盘合二为一。这使得数据写入即持久，系统崩溃时不再需要重做事务操作，消除了传统 WAL 日志中的重做日志。考虑到持久屏障的延迟可能很大，他们还提出了 NVM 组提交的方法，在 DRAM 中进行数据库数据操作，按批复制到 NVM 中进行持久化。这两种方法除了取消了重做日志外，由于回滚日志不再需要顺序，因此还取消了集中式日志，每个事务维护自己的私有日志，减少了日志的竞争。

MARS[13]是一种新的 WAL 引擎方案，其依赖于称为 EAWs 的硬件辅助原语。EAWs 允许应用程序安全地访问和修改已写的日志，日志不再是只追加的。MARS 通过使用 EAWs 消除了回滚日志，并且在提交时应用重做日志来实现数据的更新。

Arulraj 等[14]对比了传统的原地更新、写时复制更新和日志结构更新在 NVM 上的性能，并且针对 NVM 做出了相应的优化。最后结果显示，基于 NVM 的写时复制更新不适用于写密集负载；基于 NVM 的日志更新结构更像原地更新，但是多出很多开销；而基于 NVM 的原地更新效果最好。由于数据直接在 NVM 中持久化，Arulraj 等和 Pelley 等一样取消了重做日志。除此以外，他们在日志中不再记录日志的镜像，而是记录一个指向原来数据条目的指针。这种方案减少了数据冗余，进一步减少了 NVM 的写操作。

Zhang 等[15]考虑了更细粒度的写操作，一个写操作是对64位的修改，如果两个写操作分别是修改前32位和后32位，则可以合并为一个操作。如果两个操作互相抵消，则不进行更改。通过这种方法减少了日志的写操作。

以上方案除了利用了NVM的读写性能和字节寻址性，还在传统算法上进行了优化，减少了写操作。优化策略主要是消除回滚日志和重做日志中的一种或者使用其他消除数据冗余的设计。

2.3 基于异地更新的日志方案

Gao 等[16-17]用PCM作为日志缓存区，设计了PCMLogging方案，将缓存更新和隐式事务日志结合在一起，消除了显示的回滚和重做日志，DRAM中清除的脏页才会被缓存到PCM中。PCM用一个列表记录所有在PCM缓存脏页的事务，并且对于每一个脏页，记录导致该页脏的最后一个事务XID。只有该事务被提交，才会删除XID。恢复期间，为了撤销操作，PCM还需要记录所有正在进行的事务以及该事务所覆盖的页的原始版本。这种方案是一种典型的异地更新方案，提交到NVM中的数据不再覆盖原来的数据，而是写入新的位置，并且事务提交结束后删除原来的数据。但是PCM-Logging并没有从计算机结构中删除磁盘，因此从磁盘的角度看，仍然是原地更新。

Oukid 等[18]提出了SOFORT模型，对列存储模型引入NVM，每一列用一个字典描述，两个字典一一对应表示列与列的关系。MVCC数组描述事务状态，每个事务的CTS为起始时间戳，DTS为结束时间戳，DTS为无穷大说明还没有提交。对记录的更新实际上是创建元素与元素之间新的对应关系并添加在字典中，并且将原有关系的DTS设置为当前时间，表示已经被删除，但是该条目实际仍然停留在物理设备中。

Arulraj 等[19]提出了一种WBL方案，其实际上也是将更新操作转化为一个插入和一个删除操作，本质上是一种异地更新方案。

异地更新的方案因为保留了原来的数据，所以在恢复过程中省略了WAL中的redo/undo操作，可以达到很快的恢复速度，但是由于事务提交后需要回收空间，为存储空间管理带来了很大的挑战。

3 NVRC设计

3.1 NVRC设计的动机

NVM的读写性能和寿命相对于DRAM来说仍有差异，因此并不适合完全替代DRAM担当读写频繁且对读写延迟有很高要求的缓存工作。从目前的发展趋势看，NVM更适合作为持久化存储介质，在数据库系统中作为数据和日志的持久存储。

既然引入了写寿命有限的NVM作为持久存储，那么延长NVM的使用寿命就是一个必须考虑的问题。这不仅需要考虑NVM的磨损均衡，更重要的是要减少NVM的写操作，因此我们希望设计出一种能够最大程度地减少写操作的日志方案。

首先，单纯的原地更新方案无法解决“写两次”问题，即为了恢复到一致性状态，必须把被修改的数据进行备份。异地更新方案虽然避免了“写两次”问题，但是由于数据库记录的连续性，无法关注一条记录内部的变化。如果一条长记录仅修改了很少的一部分，此时异地更新会造成大量数据的重复写入。无论是传统的影子分页方案，还是PCMLLogging中的“影子记录”方案，都没有充分利用NVM的缓存行(64B)[4]写入特性来解决上述问题。

如果将存储数据的持久性设备换成NVM，则每个检查点只需要将更改的缓存行写入原数据位置就可以完成更新。我们称这种方案为“缓存行原地更新”，即记录缓存行粒度的日志并进行缓存行粒度的原数据覆盖。NVCL[3]虽然也采用了缓存行粒度来记录更新，但它将NVM作为磁盘的缓存使用，最终还需要将数据以页的粒度写回磁盘。因此，本文提出的“缓存行原地更新”策略与NVCL本质上是不同的。

通过上文分析可以看到，原地更新的数据冗余来源于被修改的数据，而异地更新的数据冗余来源于未被修改的数据。因此综合考虑，设计了NVRC。NVRC的主要思路是在“影子记录”的思想上动态地决定是采用异地更新的“影子记录”还是缓存行粒度的原地更新。具体而言，在把DRAM中的数据提交到NVM前，对修改后的记录和原来的记录进行缓存行级别的比较。如果原地更新以及其备份的I/O代价超过了“影子记录”的代价，则选择“影子记录”；否则，选择“缓存行原地更新”策略。

图1给出了WAL、“影子记录”和NVRC的区别。

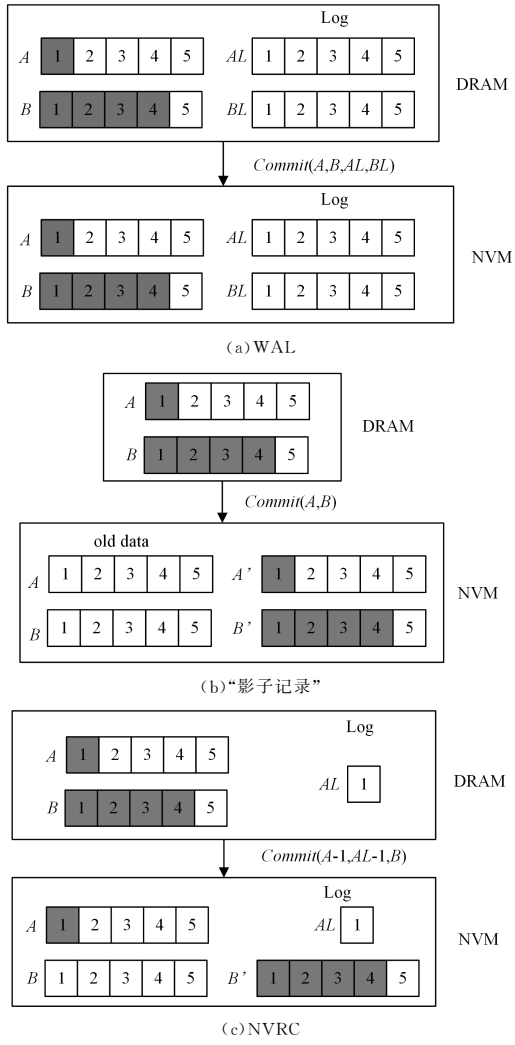

图1 3种日志方案的对比

Fig. 1 Comparison of three logging schemes

如果 A，B 两个记录各有 5 个缓存行，在 WAL 下，这两个记录在备份之后原地更新，一共需要写入 20 个 NVM 缓存行。在“影子记录”方案下，将这两个记录写入其他位置，即 A'，B'，此时写入 10 个缓存行。而在 NVRC 方案下，通过比较发现 A 中只有第一个缓存行发生了更改，如果采用缓存行的原地更新，则连同备份一共需要写入 2 个缓存行，比“影子记录”的 5 个缓存行代价更小。而 B 中有 4 个缓存行发生修改，如果采用“缓存行原地更新”，连同备份需要将 8 个缓存行写入 NVM，大于“影子记录”的代价，因此选择“影子记录”方案，最后一共写入 7 个 NVM 缓存行。该方案比 WAL 的 20 个缓存行和“影子记录”的 10 个缓存行的 I/O 代价更小，NVRC 在这种情况下具有很好的效果。

3.2 NVRC 的实现

3.2.1 NVRC 的日志结构

对于内存中每一个事务的运行过程，系统将会记录该事务修改、插入以及删除。在事务结束后，NVRC 以事务为单位构建出恢复日志。考虑到“影子记录”的方案可以拆分成一个插入和一个删除操作，因此可以将其并入原有的插入和删除中。最后每个事务的日志结构如图 2 所示，都有 5 个部分：提交标记、事务 ID、插入、删除和更改。提交标记表明该事务是否已经完成数据提交。插入部分是一个表示地址的数组，每个元素是插入记录的地址，它既可能是一个新插入的记录，也可能是一个修改后的记录。删除部分和插入一样，每个元素是一个删除记录的地址，既可能是原有的记录被删除，也可能是一个被修改的记录原来的地址。而更改部分由 alter frame 构成，每一个 alter frame 代表一个选择“缓存行原地更新”的记录。如图 2 所示，Alter frame 由 4 个部分构成，Addr 为被修改记录的地址；Record size 表示被修改记录的缓存行大小；Bitmap 是被修改的缓存行的位图，用来表示哪些缓存行被修改；Data 是被修改前的数据。

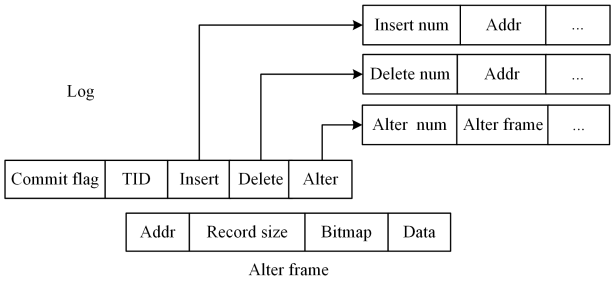

图 2 NVRC 日志结构

Fig. 2 NVRC log structure

3.2.2 NVRC 工作过程

事务结束后，根据事务过程中记录的信息做如下操作。

插入：将该记录地址放入一个插入列表。

删除：将该记录地址放入一个删除列表。

更新：对新数据和原来的数据进行缓存行级别的比较。根据代价的判断选择合适的更新方案。如果选择“影子记录”，则将更新操作转化为一个插入和删除操作，并且将地址分别放入相应的列表。如果选择“原地更新”，则构建 Alter frame。图 3 展现了一个 Alter frame 构建的过程，对于一个 10 个缓存行大小的记录，Record size 记录为 10。10 个缓存行需要 10 个 bit 位记录，1 代表被修改，0 代表没有被修改。因此 Bitmap 的大小为 2 个字节，其前 10 位 bit 有含义，后 6 位 bit 没有意义。Data 则是被修改的 4 个缓存行的原始数据，用于备份，其对应的实际数据位置由 bitmap 决定。

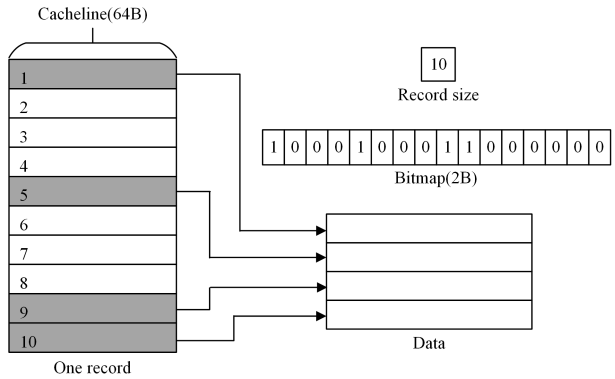

图 3 alter frame 构建过程

Fig. 3 Construction process of alter frame

提交：建出事务的日志以后，先将该日志写入 NVM，完成日志的写入以后，将数据根据所选方案写入 NVM，数据全部提交完成后，对日志 commit flag 写 1，表示事务完成提交。

恢复：RAM 崩溃时，需要通过日志对数据库中的数据进行恢复。扫描日志，对未提交成功的日志进行操作。对插入列表中的地址进行空间释放，对删除列表中的记录进行删除恢复，将更新列表中的每个记录重新写入原数据，从而使其恢复到崩溃前的一致性状态。

3.3 时间代价分析

与 WAL 和“影子记录”相比，NVRC 在一些情况下减少了对 NVM 的缓存行的写操作，这一方面提高了 NVM 的寿命，另一方面减少了对 NVM 的写操作代价，提高了运行时的性能。但是 NVRC 的性能与数据库的负载息息相关，如果数据库的更新操作对一条记录的更改范围很大，则最后选择的方案都是“影子记录”，这种情况下，不仅没有带来性能上的优势，反而引入了多余的比较过程。而如果数据库中的绝大部分更新都是对一个很长记录的很小一部分进行修改，则 NVRC 可以节省大量 NVM 的写代价。

NVRC 引入的代价主要是内存中更多的数据结构维护以及新数据和旧数据比较时的 CPU 代价。其中内存的数据结构维护可以通过增大缓存来减少对内存的访问，从而降低开销；而对于比较时的 CPU 代价，可以采用一些优化策略，如定制汇编和选择合适的比较策略等。

3.4 磨损均衡考虑

前面提到延长 NVM 的寿命可以从两方面考虑，一个是减少写操作，另一个是使 NVM 的磨损保持均衡。本文的 NVRC 实际上是引入了一种选择机制：“影子记录”或者原地更新。“影子记录”使系统可以自由选择另一个空间，以减少对原来位置的磨损，而原地更新刚好相反。

因此，如果在选择中增加对磨损的考虑，可以结合其他的 NVM 整体磨损均衡算法[17]。当一个记录多次被修改很小部分时，将不再选择原地更新，而是利用“影子记录”减少对原来位置的磨损。

4 性能验证

4.1 实验设置

实验运行在一台具有 Intel Corei3-9100 CPU 的主机上，该 CPU 拥有 256 kB 的 L1 缓存，1 MB 的 L2 缓存以及 6 MB 的 L3 缓存。计算机的内存为 8GB 的 DDR4 的 DRAM 内存。实验用增加延迟的方案来模拟 NVM 的性能，所有的读写都在 DRAM 中进行。实验根据现有的情况将 NVM 写延迟设置为 200 ns，读延迟设置为 50 ns，并且认为这样的设置已经将 NVM 相关指令的影响包含在内，即每一个缓存行的读需要 50 ns，写需要 200 ns。由于 NVRC 借鉴了影子分页的思想，因此实验以一种完全使用“影子记录”的方案作为对比，由于这种方案和 PCMLlogging 接近，故被称为 PCMLx。除此以外，用一种记录级的 WAL 日志技术作为基准参考。由于 NVRC 的设计并没有涉及查询索引等方面，为了减少数据库其他部件对实验的影响，索引采用最简单的 hash 索引，并且默认索引已经全部读入内存，无须从 NVM 中读取。但是，如果每次修改均使用“影子记录”方案，则需要对 NVM 进行索引地址的修改操作。

4.2 实验负载

YCSB 是一种被广泛使用的键值存储压测工具[20]。该键值数据库每个元组的大小约为 1 kB。实验使用相同形式的数据库，该数据库中共有 10 万个键值对，为了对齐 8 字节，key 设置为占据 16 字节的数字，value 为 10 个单元构成的字符串，每个单元有 100 个字节。为了模拟与数据库相对应的内存，实验设置内存中只能容下 10%的键值对，即 1 万对键值对，并且访问缺失时使用 LRU 算法替换数据。由于查询不是实验关注的重点，且 NVRC 设计更偏重于更新操作的优化，实验设计了一个具有 10 万次更新操作的负载，其中 90%的更新是对 10%的键值对的操作。因为每一个 value 的大小为 1000 B，约占 16 个缓存行，所以具体的实验运行中，一旦发现 8 个缓存行被修改，即停止比较，选择“影子记录”。根据 NVRC 的设计，数据修改的不同会导致不同的策略选择，对性能的影响很大，因此为了更全面地了解 NVRC 的优势与缺点，我们设置了 3 种更新操作负载。

1)均匀修改：以 100 字节单元为单位，每次更新修改 n 个单元($1<n<=10$)，而修改 n 个单元的比例都是 10%。也就是说，有 10%的更新修改 1 个单元，即连续的 100 个字节；同样也有 10%的更新修改了全部的单元，即 1000 字节。

2)少数修改：为了检验 NVRC 在极端情况下的优势，每次更新只修改 1 个单元，这样可以保证所有的更新都选择“缓存行原地更新”的策略。

3)多数修改：为了检验 NVRC 在极端情况下的劣势，每次更新都修改全部的单元，这样可以保证所有的更新都选择“影子记录”策略。

4.3 实验结果与分析

由于数据的修改情况并不影响 PCMLx 和 WAL 的性能，因此不将这两种方案针对不同修改情况分开评估。图 4 和图 5 给出了 3 种方案在 5 种情况下的运行结果，其中 NVRC_1 代表均匀修改的负载，NVRC_2 代表少数修改的负载，NVRC_3 代表多数修改的负载。

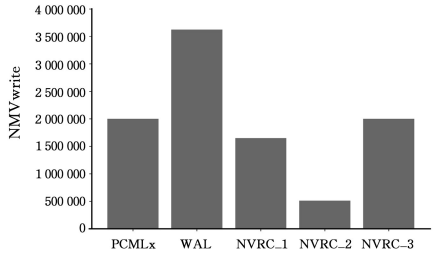

图 4 NVM 的写次数

Fig. 4 Number of NVM writes

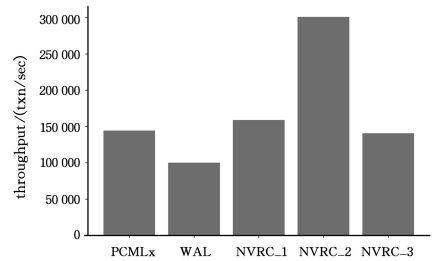

图 5 更新吞吐量

Fig. 5 Updating throughput

4.3.1 NVM 写次数

图 4 为 5 种情况下对 NVM 的写操作次数，每次写的单位为连续的 64 B。可以看出，WAL 由于数据冗余较大，需要对 NVM 进行大量的写操作；PCMLx 因为采用“影子记录”，相比 WAL 减少了接近一半的写操作；而 NVRC 在均匀修改的情况下由于进一步减少了数据冗余，相比 WAL 减少了 54%的写操作，相比 PCMLx 减少了 17%的写操作。在少数修改的情况下，NVRC 对 NVM 的写操作更是降到了极低的水平。而对于多数修改的极端情况，由于 NVRC 最后全部采用与 PCMLx 相同的方案，因此它们的写次数基本上是一样的。可以看出，在减少 NVM 的写次数、降低 NVM 的磨损、延长其寿命方面，NVRC 相比 PCMLx 和 WAL 都有较好的效果。

4.3.2 更新吞吐量

图 5 所示为 5 种情况下的吞吐量结果，显示了 NVR 对 NVM 上更新性能的影响。同样，由于 WAL 的写入操作太多，严重影响了数据库运行的性能，而 NVRC 在均匀修改的情况下由于减少了写次数，降低了很多 I/O 代价，因此，结果显示 NVRC 的吞吐量相比 PCMLx 提高了 10%，相比 WAL 提高了 59%。NVRC 在少量修改的情况下减少了更多的 I/O，因此其吞吐量也有了很大的提高。但是在多数修改的情况下，NVRC 相比 PCMLx 并没有减少写次数，反而引入了比较的开销，因此其吞吐量相比 PCMLx 下降了 3.5%，虽然吞吐量有所下降，但是对性能并没有太大影响。这是因为比较两个不同的缓存行往往不需要比较全部数据，所以相对于比较两个相同的缓存行代价更低。因此，一个记录如果修改得太多，其比较时间也是非常短的，而一个记录若修改得不多，则几乎需要比较每一个字节，其时间开销更大，但是可以通过更少的 I/O 来弥补。

结束语 本文分析了 NVM 数据库日志研究的必要性，

并对目前的研究现状进行了总结。然后基于已有的研究，结合 NVM 的特点，提出了一个新的数据库日志方案 NVRC。NVRC 设计了新的日志结构，并且提出了融合“影子记录”和“缓存行原地更新”的日志方案，可以根据代价分析，动态地选择原地更新还是异地更新策略。我们采用仿真的方式基于 YSCB 负载对 NVRC 进行了实验验证，并与 WAL 和 PCMLx 进行了对比。结果表明，在修改较为均匀时，NVRC 不仅具有比 WAL 和 PCMLx 更少的 NVM 写操作，同时也具有更高的更新性能。

在未来的工作中，我们将首先基于 NVRC 方案完善数据库的其他组件，如内存和 NVM 空间的管理系统、适合 NVRC 的索引算法以及对并发性的支持等。其次，我们将考虑在一个开源的数据库管理系统上以及真实的 NVM 设备上实现 NVRC 方案。最后，我们将继续探讨引入 NVM 后不同结构下的数据库变化，如 NVM 和 DRAM 的混合内存结构。

参 考 文 献

[1] KIM N, SONG C, CHO W, et al. LL-PCM: Low-latency phase change memory architecture[C]// DAC. 2019: 14.

[2] XIA F, JIANG D, XIONG J, et al. A survey of phase change memory systems [J]. Journal of Computer Science and Technology, 2015, 30(1): 121-144.

[3] ZHANG M, YAO X, WANG C L. NVCL: Exploiting NVRAM in cache-line granularity differential logging [C] // NVMSA. 2018: 37-42.

[4] WU Z L, JIN P Q, YUE L H, et al. A survey on PCM-based big data storage and management [J]. Journal of Computer Research and Development, 2015, 52(2): 343-361.

[5] LUO Y P, JIN P Q. Optimizing Join Algorithms for NVM/DRAM-Based Hybrid Memory Architecture[J]. Chinese Journal of Computers, 2020, 43(6): 1069-1085.

[6] OUKID I, KETTLER R, WILLHALM T. Storage class memory and databases: Opportunities and challenges [J]. it-Information Technology, 2017, 59(3): 109-115.

[7] MOHAN C, HADERLE D, LINDSAY B, et al. ARIES: a transaction recovery method supporting fine granularity locking and partial rollbacks using write-ahead logging [J]. ACM Transactions on Database Systems, 1992, 17(1): 94-162.

[8] GRAY J, MCJONES P, BLASGEN M, et al. The recovery manager of the System R database manager [J]. ACM Computing Surveys, 1981, 13(2): 223-242.

[9] FANG R, HSIAO H I, HE B, et al. High performance database logging using storage class memory [C] // ICDE. 2011: 1221-1231.

[10] KIM K, LEE S W, MOON B, et al. IPL-P: In-page logging with PCRAM [J]. Proceedings of the VLDB Endowment, 2011, 4(12): 1363-1366.

[11] LEE S W, MOON B. Design of flash-based DBMS: an in-page logging approach[C]// SIGMOD. 2007: 55-66.

[12] PELLEY S, WENISCH T F, GOLD B T, et al. Storage management in the NVRAM era [J]. Proceedings of the VLDB Endowment, 2013, 7(2): 121-132.

[13] COBURN J, BUNKER T, SCHWARZ M, et al. From ARIES to MARS: Transaction support for next generation, solid-state drives[C]// SOSP. 2013: 197-212.

[14] ARULRAJ J, PAVLO A, DULLOOR S R. Let's talk about storage & recovery methods for non-volatile memory database systems[C]// SIGMOD. 2015: 707-722.

[15] ZHANG W Z, LU K, LUJÁN M, et al. Write-combined logging: an optimized logging for consistency in NVRAM [J]. Scientific Programming, 2015, 2015: 398369.

[16] GAO S, XU J, HE B, et al. PCM Logging: reducing transaction logging overhead with PCM[C]// CIKM. 2011: 2401-2404.

[17] GAO S, XU J, HÄRDER T, et al. PCM Logging : Optimizing transaction logging and recovery performance with PCM [J]. IEEE Transactions on Knowledge and Data Engineering, 2015, 27(12): 3332-3346.

[18] OUKID I, BOOSS D, LEHNER W, et al. SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery [C] // DaMoN. 2014: 1-7.

[19] ARULRAJ J, PERRON M, PAVLO A. Write-behind logging [J]. Proceedings of the VLDB Endowment, 2016, 10(4): 337-348.

[20] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]// SoCC. 2010: 143-154.

FAN Peng-hao, born in 1998, postgraduate. His main research interests include database technology for NVM and so on.

JIN Pei-quan, born in 1975, Ph.D, associate professor, is a senior member of China Computer Federation. His main research interests include database and big data.