浙江工业大学

本科毕业设计外文翻译

（2013 届）

论文题目　主内存数据库系统概述

作者姓名	陈佳鹏
指导教师	陈　波
学科 (专业)	软件工程
所在学院	计算机科学与技术学院
提交日期	2013 年 06 月

主内存数据库系统概述

摘要： 内存驻留数据库系统（MMDB）将数据存储在其主要物理内存中并提供非常高的速度对数据进行访问。传统的数据库系统主要针对磁盘存储机制的具体特点来进行优化。相反的是，内存常驻系统对结构和组织数据使用不同的优化从而使其可靠。本文概述的主要是内存常驻的优化并简要讨论了一些常驻内存系统的设计和实施。

关键字： 访问方式，应用编程接口，事务提交处理，并发控制，数据聚类，数据表示，主内存数据库系统（MMDB），查询处理，数据恢复

一、介绍

在主内存数据库系统（MMDB）中，数据常驻在物理内存中；在传统的数据库系统（DRDB）中，数据储存在磁盘上。在 DRDB 中，磁盘上的数据可能会被缓存到内存中以便被访问；在 MMDB 中，驻留在内存中的数据可能在磁盘上有一个备份。所以在这两种情况下，一个给定的对象在内存和磁盘上都有备份。关键的区别是，数据在 MMDB 中的那个主要备份会永久驻留在内存中，这具有重要的含义。

随着半导体内存的芯片密度增加而且价格更便宜，这使它可现可以在内存中储存更大的数据库，使得 MMDB 的变成实。由于数据可以直接在内存中访问，相比于 DRDB，MMDB 能提供更好的响应时间和事务吞吐量。对于必须在指定的期限前完成任务的实时交易应用程序，这是尤其重要的。

一台电脑的内存和磁盘有着明显不同的属性，而这些差异对数据库系统的设计和性能有着深刻的意义。尽管这些差异是众所周知的，但是值得简要的回顾一下。

1）访问磁盘数据的时间是访问内存所需时间的指数倍。

2）主内存通常是不稳定的，而磁盘存储则不是。但是，花费一定的成本去构建稳定的主内存也是可行的。

3）磁盘每次访问有一个高且固定的成本，且不依赖于期间检索访问的数据量。出于这个原因，磁盘是面向块的存储设备，主内存则不是。

4）由于在磁盘上，连续访问比随机访问更快，数据在磁盘上的布局比在内存上的布局更加重要。而连续访问在内存中并不重要。

5）内存通常可以直接被处理器访问，而磁盘则不是。对于软件错误而言，这使得在内存上的数据比在磁盘上的数据更加脆弱。

从并发控制到应用程序接口，以上的差异几乎对数据库管理的每一个方面都有影响。在本文中我们将讨论这些影响并且将会简单地概述一些最近已被设计或实现的 MMDB。然而，在开始之前，先解决常见的关于 MMDB 的三个问题。

假设将整个数据库放入到内存中是否合理？是的，对于一些应用程序来说，在某些情况下，整个数据库有大小上限，或者数据库膨胀的速率小于内存容量的增长速率。例如，数据库的大小和公司的员工数或客户数成比例，不管这家公司有多成功，对于每个员工或客户，我们有理由去期望内存能储存几百乃至几千字节的数据。对于一些事实的应用程序，数据必须被储存在内存中以满足事实处理的这个限制，所以数据库大小一定不能超过可用的内存大小。下面举几个事实处理的应用程序：电信，雷达跟踪，安全贸易等。

但是，在一些明显的情况下，永远不适合将整个数据库放到内存中，比如一个包含全球卫星图像数据的应用程序。所以在这种情况下，DRDB 还将发挥重要的作用。尽管如此，即使在这样大的应用程序中，也很容易找出两种类型的数据：经常被访问的“热数据”（通常是些体积小、有严格时间要求的数据），偶尔被访问的“冷数据”（通常是些体积大、各种各样的中间层数据）。

在这种情况下，可能要将数据分割到一个乃至更多个逻辑数据库中并将最“热”的数据储存到内存中。至此，我们有了一个数据库的集合，一些是 MMDB，其余的是 DRDB。数据库系统可能是完全不相交的，使得应用程序访问它们时相当于访问一个失去同盟的数据库系统，他们将访问失去联邦数据库系统；或者它们可能紧密集成，对于那些经常被访问的数据改变时，有功能自动的将数据从一个数据库系统迁移到别处。迁移数据便不简简单单的是对一些数据进行缓存。通过缓存，对象的临时副本可以创建在一个不同的储存级别上。通过数据迁移，对象被移动到不同的管理系统中，其结构和访问途径可能会发生变化。

很多应用程序的部分数据会自然的增长，例如，在银行业务中，账户记录通常是“热数据”，客户记录和历史记录通常是“冷数据”；在一个通话选择应用程序中，通话线路表是“热数据”，客户的月账单是“冷数据”。

IMS，作为最早的数据库系统之一，认识到数据访问的不同性，多年来提供了两套系统：内存中的数据用 Fast Path，其余的数据用 IMS。最近一篇由 Stonebraker 发表的文章中讨论了一些包含在多级别数据库系统和数据迁移中的问题。在本篇文章的余下部分中，我们的“数据库”指代的是永久在内存中由 MMDB 管理的那部分数据。

MMDB 和拥有一大块缓冲区的 DRDB 有什么区别？如果 DRDB 的缓冲区够大，数据的备份会一直驻留在内存中。尽管这样的一个系统会有良好的性能，但是它不会利用到所有的内存。例如，即使数据在内存中，索引结构也会被设计成通过磁盘访问。同时，应用程序可能必须通过缓冲管理程序来访问数据，就好像数据在磁盘上一样。又例如，每次一个应用程序希望访问一个特定的元组，就要计算它在磁盘上的物理地址，然后缓冲管理程序会被调用去检查相应的数据块是否在内存中。当数据块在内存中的时候，这个元组会被拷贝到应用程序的缓冲区中。如果这条记录一直在内存中，通过内存地址来引用它将会变得更加有效率。

以上我们阐述过的只是可能的内存优化方案中的一个，其余的将会在第二章节阐述。一些 DRDB 和一些含面向对象储存系统已经意识到，通过大缓存的帮

助，部分数据可以经常驻留在内存中，并且已经实现了一些关于 MMDB 的内存优化方案。例如，一些新系统将一个元组或对象转化成内存中的一个替代物，并且给应用程序一个直接指向它的指针。随着 DRDB 提出越来越多的优化方案，DRDB 变得越来越接近 MMDB。在不久的将来，我们希望 MMDB 和 DRDB 的差别消失；一部分数据将会永久驻留在内存中并应该被相应的管理，任何优秀的数据库管理系统会意识到并利用这个道理。

我们能通过特殊的硬件来使得内存变得稳定和可靠吗？自从设计 MMDB 变得越来越简单且其性能会更加提高开始，这就是个十分吸引人的假设，对于这个问题，没有确切的答案。内存只是一个储存介质，可以通过一些技术使其变得更加可靠，例如：通过电池支持的内存板，永不断电的电源，错误检测并能修复的内存，三重模块冗余。这也仅仅是减少介质出错的可能性而已，必不能讲其变成零。因此，一般在磁盘上必须对数据库有个备份。对于 DRDB 来说，也需要相应的或在其他磁盘上的备份。

由于介质出错的可能性在减少，备份的次数也可以减少，这些备份表现出的意义也在减少。例如，我们可以利用优秀的磁盘来做到一周进行一次备份即可。对整个数据库扫描和备份的开销将显得不再那么重要。由于常驻于内存数据的关系，一些因素会促使备份次数的上升。

1）由于处理器对内存可以直接访问，其对操作系统的错误而言，内存就更易受攻击。所以即使硬件十分可靠，当系统出问题的时候，内存中的内容就会定期的丢失。

2）当磁盘出问题的时候，在不影响其他磁盘的内容的情况下也能被修复。恢复之后，只需要将一部分数据库从备份中恢复，同时，在恢复过程中，数据库中的其余部分也能被访问。当内存板出现问题的时候，通常整个机器必须关机，这会丢失整个数据库。由于数据恢复会消耗大量时间，激活最近一次的备份是个可取的方法。

3）电池供电的内存或有永不断电的供电支持（UPS），这些都属于“活跃”的设备，使得数据更加容易丢失。磁盘属于“消极”的设备，为了储存数据不用特意做些工作。UPS 会消耗掉可燃气并且提高温度，电池也会漏电或失效。

综上所述，除非能充分相信内存的可靠性，应该相对性的经常给内存数据备份，备份数据将会起到重要的作用。由于磁盘写入比内存写入开销更大，在常规系统中，备份的开销要比等量的磁盘到磁带备份开销显得更加重要。

二、内存数据的影响

我们已经提出理由说明数据库系统在管理常驻内存数据时有别于常规系统。在接下去的章节中，我们将会讨论常驻内存对数据库管理系统产生的影响。

2.1 并发控制

由于访问内存比访问磁盘快很多，我们希望在主内存系统中事务能更快处理。在使用锁进行并发控制的系统中，锁不能被占用太久，当数据常驻于磁盘时，

锁的竞争可能显得没那么重要。(这里我们专注于基于锁的并发控制,因为这在实践中使用的最多并且已经在 MMDB 的原型中使用。)

那些通过给小块数据(属性域或记录)加锁的系统就是靠此来减少锁的竞争。因为数据常驻在内存,如果锁的竞争已经不频繁,给小块数据加锁的主要优势实际上已经被忽略了。出于此原因,大块数据(例如关系网)最适合作为常驻内存的数据。

在极端情况下,加锁的小块数据会被当做成整个数据库。这相当于串行执行的事务。串行事物处理是很可取的,因为并发控制的开销几乎完全被忽略了。此外,不通过 CPU 缓存而直接从原地址获得数据的次数在大幅度减少(每次一个事务暂停并等待锁时,一个新的事务正在运行,CPU 缓存中的内容必须被更新。在串行执行的情况下,对于每个事务只需要一个 CPU 缓存更新器)。在高性能的计算机中,缓存刷新相当于上千条指令,获益非常显着。然而长期事务存在时,串行事务可能是不实际的(例如,对话型交易)。出于公平,运行长期事务时应该以某种方式同步运行短期事务。此外,多处理器系统可能需要某种形式的并发控制,即使都是短期事务。

锁机制的实现也要根据锁定常驻内存的对象而优化。在常规系统中,通过一个包含当前被锁住对象条目的哈希表来实现锁,这些在磁盘上的对象本身不包含锁的信息。如果对象在内存中,我们可能要拿出几位来表示锁的状态。

举例而言,我们只对特别的锁考虑,如果第一位是 1 则对象被锁,反之。如果前两位都是 1,说明有一个乃至更多的事务在等待,这些事务的属性被储存在一个常规的哈希表中。如果一个事务希望锁住一个对象,它首先检查其第一位,如果是 0,就置为 1,直到处理结束。之后如果另外一个事务想要在这个对象等待,它便将其第二位置为 1 并将自己加到这个对象的等待队列中。

当处理中的那个事务将第一位置为 0 时,它检查第二位是否是 1。如果不是,说明没有其他事务在等待,结束。如果是,它必须通过常规过程唤醒一个正在等待中的事务。很明显,我们忽略了很多细节问题,但是关键点在于迄今为止,最有可能的情况是一个事务在其他事务等待之前锁住,处理,释放一个对象。在这种情况下,锁住和释放能在最少的机器指令下完成,避免了对整个哈希表的查找。当然必须考虑一个额外的空间开销。对于那些十乃至更多字节的典型数据,几比特的开销就不显得那么重要了。

2.2 数据表示

主内存数据库也能利用有效指针指向数据表示,相关的元组可以被表示成一些指向数据的指针集合。当较大的值在数据库中出现很多次的时候,指针的空间

能被有效的利用，因为实际的值只需要储存一次。由于长度可变化的数据能被表示成指向堆内数据的指针，指针简化了处理可变长度字段的问题。

三、系统

目前，已经提出或实现部分管理内存常驻数据的数据库管理系统，包括纸上设计（MM-DBMS，MARS，HALO）到商业系统测试平台。由于空间有限，我们就做一点短暂的描述。此外，我们将讨论的重点放在“这些系统是如何解决由驻留在内存中的数据引发的问题？”上。

3.1 MM-DBMS

MM-DBMS 系统由威斯康星大学设计，和 OBE 类似，它实现了关系型数据模型并在数据表示和访问方法上广泛的使用指针。可变长度的特征值被表示成指向堆的指针，临时的关系网通过指向数组的指针实现。索引结构的指针直接指向被索引的元组，并不存储实数据，通过一变种线性哈希来索引乱序的数据，而通过 T 树来访问有序的数据。

出于恢复数据的目的，内存被分割成较大的几块，是用来和磁盘上数据库备份之间传输的单元。提交处理包含稳定的内存（记录日志）和一个单独的恢复处理器，恢复处理器组根据各自引用的记录块可以独立的进行恢复工作。当块内得到了一定数量的更新之后，恢复处理器就会对其进行检查。在检查的过程中会设置一个锁，以确保每个在磁盘上的块保持事务的一致性。备份失败之后，块需要被重新拿回内存并通过其日志记录组做日志记录。

MM-DBMS 对大数据颗粒（例如，整个关系网）使用两面锁做并发控制。

3.2 MARS

MARS MMDB 由南美以美大学设计，利用一对处理器来对内存常驻数据做快速的事务处理。

MARS 系统包括一个数据库处理器和一个恢复处理器，它们能访问一个包含数据库的不稳定主内存和一个稳定的主内存。恢复处理器能访问磁盘的日志记录和数据库的备份副本。

数据库处理器负责事务的执行到事务的提交，除非提交了更新的事务，这些更新不会影响到主数据库的那个备份副本。相反，数据库处理器会记录下稳定内存中的更新。如果更新中的事务中止了，被记录下的更新就被丢弃。

为了提交一个事务，恢复处理器从稳定内存中拷贝其记录下的更新记录到数据库中，这些记录被拷贝到一个稳定的日志缓冲区中。恢复处理器负责刷新整个日志缓冲区到记录日志的磁盘中，同时也定期的做一些检查，将数据库改变的部分刷新到备份中。

也是对大数据颗粒（例如，整个关系网）使用两面锁做并发控制。

四、总结

在《The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time》这篇文章中阐述了“每隔 5 分钟就被引用至少一次的数据应该作为内存常驻数据”这个观点。通过对内存和磁盘数据访问的花费分析，提出了“5 分钟”这个数值。重要的一点是，和每秒对磁盘访问的开销相比，主内存每个字节成本在下降，得到结果的时间增加了，也就是说，未来的“5 分钟原则”要变成“10 分钟原则”了。随着内存越来越便宜，将更多数据永久的保存在内存中是个有效的途径。这意味着在不久的将来，内存数据库系统会变得更加常规化，同时本文中讨论过的机理和优化将变得非常普遍化。

参考文献

[1] Ammann A, Hanrahan M, Krishnamurthy R. Design of a memory resident DBMS[C]. IEEE COMPCON, 1985.

[2] Bitton D, Hanrahan M, Turbyfill C. Performance of complex queries in main memory database systems[C]. Proceedings of the Third International Conference on Data Engineering, 1987:72--81.

[3] Copeland G, Keller T, Krishnamurthy R, et al. The case for safe RAM[C]. Proceedings of the 15th international conference on Very large data bases, 1989:327--335.

[4] Copeland G, Franklin M, Weikum G. Uniform object management[J]. Advances in Database Technology—EDBT'90, 1990:253--268.

[5] DeWitt D J, Katz R H, Olken F, et al. Implementation techniques for main memory database systems[M]. Vol. 14.[S.l.]: ACM, 1984.

[6] Eich M H. A classification and comparison of main memory database recovery techniques[M]. [S.l.]: IEEE Press, 1989.

[7] Eich M H. MARS: The design of a main memory database machine[J]. Database Machines and Knowledge Base Machines, 1988, 43:325.

[8] Garcia-Molina H, Salem K. High performance transaction processing with memory resident data[C]. International Workshop on High Performance Transaction Systems, 1987.

[9] Gawlick D, Kinkade D. Varieties of concurrency control in IMS/VS fast path[J]. Database Engineering Bulletin, 1985, 8(2):3--10.

[10] Gray J, Putzolu F. The 5 minute rule for trading memory for disc accesses and the 10 byte rule for trading memory for CPU time[C]. ACM SIGMOD Record, 1987，16:395--398.

[11] Reuter G J, Gray J. Transaction Processing: Concepts and Techniques,>[J]. MorganKaufmann, San Mateo, CA, 1993.

[12] Gruenwald L, Eich M H. MMDB reload algorithms[C]. ACM SIGMOD Record, 1991，20:397--405.

[13] Hagmann R B. A crash recovery scheme for a memory-resident database system[J]. Computers, IEEE Transactions on, 1986, 100(9):839--843.

[14] Kim M Y. Synchronized disk interleaving[J]. Computers, IEEE Transactions on, 1986, 100(11):978--988.

[15] Lehman T J, Carey M J. Query processing in main memory database management systems[C]. ACM SIGMOD Record, 1986，15:239--250.

[16] Lehman T J, Carey M J. A study of index structures for main memory database management systems[C]. Conference on Very Large Data Bases, 1986，294.

[17] Lehman T J, Carey M J. A recovery algorithm for a high-performance memory-resident database system[C]. International Conference on Management of Data: Proceedings of the 1987 ACM SIGMOD international conference on Management of data: San Francisco, California, United States, 1987，27:104--117.

[18] Li K, Naughton J F. Multiprocessor main memory transaction processing[C]. Proceedings of the first international symposium on Databases in parallel and distributed systems, 2000:177--187.

[19] Patterson D A, Gibson G, Katz R H. A case for redundant arrays of inexpensive disks (RAID)[M]. Vol. 17.[S.l.]: ACM, 1988.

[20] Pucheral P, Thévenin J M, Valduriez P. Efficient Main Memory Data Management Using the DBGraph Storage Model[C]. Proceedings of the 16th International Conference on Very Large Data Bases, 1990:683--695.

[21] Reuter A. Performance analysis of recovery techniques[J]. ACM Transactions on Database Systems (TODS), 1984, 9(4):526--559.

[22] Salem K, Garcia-Molina H. DISK STRIPING t[C]. International Conference on Data Engineering, February 5-7, 1986, Bonaventure Hotel, Los Angeles, California, USA., 1986:336.

[23] Salem K, Garcia-Molina H. Checkpointing memory-resident databases[C]. Data Engineering, 1989. Proceedings. Fifth International Conference on, 1989:452--462.

[24] Salem K, Garcia-Molina H. System M: A transaction processing testbed for memory resident data[J]. Knowledge and Data Engineering, IEEE Transactions on, 1990, 2(1):161--172.

[25] Stonebraker M. Managing persistent objects in a multi-level store[C]. International Conference on Management of Data: Proceedings of the 1991 ACM SIGMOD international conference on Management of data: Denver, Colorado, United States, 1991，29:2--11.

[26] Whang K Y, Krishnamurthy R. Query optimization in a memory-resident domain relational calculus database system[J]. ACM Transactions on Database Systems (TODS), 1990, 15(1): 67--95.