

# Flash-based DBMS

尹少宜 (Mobile 组)

## 1. 引言

1956年，第一块硬盘诞生。从此，人类存储数据的方式被改写。那些被堆积成山的企业报表所淹没的人们看到了希望，但是，当他们兴高采烈地把报表“数字化”之后突然又眉头紧锁——如何才能迅速地从数字报表中查找记录？如何在报表之间建立关联？如何让计算机利用已有数据生成新的报表？是最初的数据库人帮他们解答了这些问题，一方面将企业数据抽象为关系模型，一方面又利用磁盘的性质给出关系数据库的实现方案。企业管理人员终于如愿以偿地使用了硬盘来管理自己的数据，这也便成就了 Oracle、DB2、SQL Server 等今日的辉煌。

1969年，互联网诞生。从此，人类传输和共享数据的方式发生了根本性变化。广大拥有 PC 机的人们欢欣鼓舞，然而，当他们放肆地把自己的数据共享又从网上获取到大量信息之后开始迷茫——在浩瀚的网页数据中，如何才能瞬间得到自己最想要的信息？又如何才能让自己发布的信息在任何时候都能被需要的人看到？聪明的数据库工作者知道传统的关系数据库不能解决这个问题，人们需要的只是一个面向网页的搜索引擎，于是他们根据搜索目的和网页特征，用新的方法把数据存储到磁盘上。人们开始从漫无目的的网上冲浪转变为一针见血的网络搜索，Google 的神话广为传唱。

从数据库的辉煌到 Google 的神话，在瞬间颠覆了人们的思考方式，而未曾改变的是，磁盘一直都被用作大量数据的存储介质。然而，人类的脚步没有停止，人们采集和使用数据的方式在日复一日地发生着变化：人们不满足于在固定的位置使用 PC 机，于是发明了便携式的计算设备；人们不愿意使用陈旧的人工方式来控制工业机器，于是发明了嵌入式控制器……即便如此，人们仍不满足于只有这样的计算和控制，而是希望随时随地将收集的数据在瞬间保存、将需要的数据在瞬间获取，于是便发明了便携式的存储设备和嵌入式的存储器。这时，磁盘再也经不起考验了——它的机械特性暴露无遗：体积太大、读写周期太长、工作温度范围太小、噪声太大、抗震性太差、耗电量太高等等，都使其无法胜任移动和高温等环境下的数据存储任务。于是，1988年，一种叫做 Flash 的永久性存储介质诞生了，它读写速度快、抗震能力强、体积小、存储密度大、噪声小、耐高温，因此其应用开始广泛而迅速地渗透到磁盘所不能及的各个领域。

Flash 的种类有很多，而目前最适合用作大量数据存储介质的当属 NAND 型 Flash。喜欢随时随地管理数据的人们当然也会喜欢大容量的移动存储介质，然而他们到底要管理什么样的数据？对数据要进行什么样的操作？关系数据库是否仍能满足其需求？而这些数据应该如何

Flash 上存储？基于磁盘的数据库实现技术在 Flash 上是否还能行得通？问题又回到了最初——什么数据，怎样存储，如何查询？好奇的数据库人又开始寻找新的答案。

## 2. NAND Flash 介绍

和磁盘类似，NAND Flash 读写数据的基本粒度为页（page），然而读、写数据所需的时间却不一样，一般写入时间是读出的 3 到 10 倍。此外，Flash 不允许数据的直接覆盖写，必须首先擦除旧有数据才能写入新的，而擦除的粒度为块（block），通常一个块包含有 64 个页。擦除的速度是很慢的，而且每一块的擦除次数也有限，平均为 100 万次。下面将给出一个例子来具体说明 NAND Flash 的特点。图 1 是三星 K9K8G08U0M 芯片的 Flash 组织结构图。

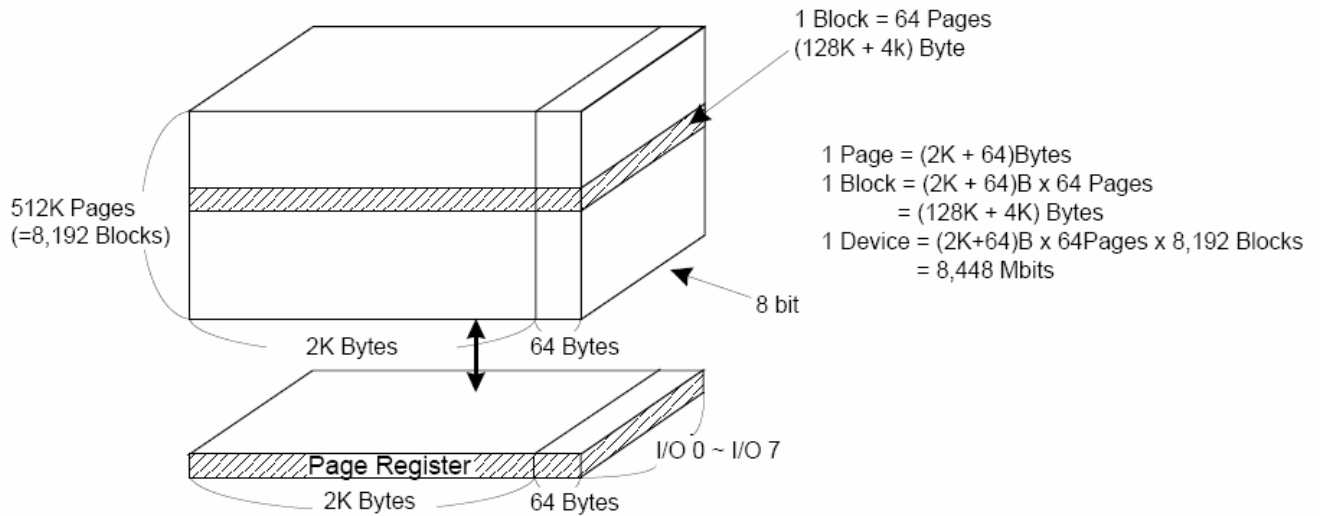


图 1: NAND Flash 组织结构图

从图中可以看出，整个 Flash 的容量为 8448Mbit，其中含有 8192 个物理块，每块包含 64 个页，每页除了 2K 的数据区之外，还有 64 字节的额外空间用于存放错误校验码等信息。除了 Flash 的主体，我们还可以看到一个一页大小的数据寄存器。事实上，每一次 Flash 的读写操作都需要使用这个寄存器来完成。例如，读操作实际上分成两个阶段，首先是把指定地址中的整页数据载入到寄存器中（时间为 20 微秒），然后再从寄存器输出数据，可以连续输出，也可以根据指定偏移量随机输出，且随机输出的次数不限（每个字节的输出周期为 20 纳秒）；而写操作也分为两个阶段，第一阶段是把数据从外界输入到寄存器，可以整页输入，也可以随机输入，且随机输入的次数不限，每字节的输入周期也是 20 纳秒，第二阶段便是把寄存器中的数据固化到 Flash 上，时间为 200 微秒，每一页原则上只能有一次固化操作，然而大部分 NAND Flash（如本例）允许在同一页中的几个片断按照先后顺序分几次写入，本例中为 4 次，这种情况叫做 **partial page programming**。

除了 partial page programming, NAND Flash 还有另外一种有意思的操作, 也和数据寄存器相关, 叫做 copy-back。就是说, 当某个数据页需要被复制到新的位置时, 只需将其载入到数据寄存器中, 然后根据接收到的目标地址将该页数据固化到 Flash 上新的位置, 这样就省去了整页数据在寄存器和 RAM 之间的输入输出过程。另外, 在数据被载入到寄存器之后, Flash 允许程序使用随机输入数据的方式改写该页的部分内容, 然后再固化到目标位置。

### 3. 基于 NAND Flash 的数据库系统研究

在引言中, 我们曾提到 Flash 的应用环境广泛, 而不同应用环境下的数据可能有不同的特征, 因此对数据库功能的要求也各不相同。事实上, 普适计算环境下的数据建模本身也是一个值得研究的问题, 而这里, 我们将重点针对某一种模型下的数据库来研究其在 Flash 上的实现问题。关系数据库模型不可能在任何环境下都适用, 然而能够使用关系数据库的例子却比比皆是, 例如公司职员将重要的客户和产品资料保存在自己的 PDA 上, 医生和病人将病历资料存放在带有智能卡的 USB key 上等等。因此, 我们不妨把关系数据库在 NAND Flash 上的实现作为研究的第一步。而这其中, 值得研究的问题又有很多, 主要包括:

#### 3.1 Flash-based DBMS 不同于磁盘数据库的评价指标

由于 I/O 瓶颈是用磁盘管理数据的主要难题, I/O 次数越少, 数据库的性能就越好, 因此传统数据库实现的目标也十分明确, 就是尽可能减少 I/O 代价。存储、索引、查询等的设计和实现目标也都是最小化 I/O 次数。然而, flash 与磁盘具有完全不同的性质, 例如读、写速度不等, 但都快于磁盘; 重写前要擦除; 擦除次数有限等。针对这些性质, 用单纯的 I/O 次数来评价系统是否仍然合理? 至少, 既然输入 (I) 和输出 (O) 的代价不等, 在新的指标评测中就应该考虑区别对待了。此外, Flash 大多用于资源受限的环境中, 那么内存使用情况、电源消耗情况等是否也应该作为更加重要的指标来考虑, 从而影响整个系统的设计? 我们的研究就是要将这些指标分析得更加透彻, 归纳出哪些指标是由 Flash 本身决定的, 具有广泛约束力, 而哪些指标是由特殊硬件环境或者软件应用决定的, 分析这些指标间的相互关系, 并最终给出在不同指标限制下的不同实现方案。需要解决的问题可概括如下:

- 系统整体性能与 Flash 读、写、擦等的次数之间有怎样的定量关系?
- 系统电源消耗与读、写、擦次数之间有何定量关系?
- 通常可以通过增加内存来提高系统性能, 或者为了节省内存而降低系统性能, 那么内存占用和性能优化二者之间如何权衡? 是否可以将系统目标设定为: 在任何给定大小的内存限制下, 系统都能够最大化利用资源从而使得性能最优? 也就是说, 系统实现不依赖于内存大小, 同时又能够最大化地利用已有内存。

## 3.2 Flash-based DBMS 的存储和索引模型（以及相应的垃圾回收方案）

从对研究现状的分析中可以看出，使用传统的以“块”为单位的存储方式会带来 Flash 的大量浪费和系统性能的下降。而使用日志式追加的存储方式又会使得数据记录变得无序，降低查找效率。怎样有效地组织 Flash 上的数据使得 Flash 资源在数据库应用环境下得到最合理的利用是一个十分有意义的研究问题。当然，这个问题本身也包括了索引结构的设计。经过深入分析，我们发现传统的索引结构直接用于 Flash 之上存在非常严重的问题，并将这些问题进行了归纳总结，发现了 Flash 上索引结构设计的瓶颈所在，然后正在试图寻找新的方案来解决这些问题。由于 Flash 在重写前需要擦除这一特性，无论使用何种存储和索引方案，都必须考虑垃圾回收和损耗平衡的问题，而如何解决这些问题也将对系统最终的性能产生巨大的影响。也就是说，使用不同的垃圾回收策略可能导致系统性能的巨大差异。上述问题可总结为：

- 在 Flash 上如何组织记录和索引？具体来说，就是数据库一个表中原始记录和索引条目分别以何种方式存储在 Flash 上？不同的表之间的数据存储位置又有什么关系？同一个表的不同索引结构之间的存储有何关联？
- 上述结构如何维护？也就是在有数据库插入、删除和更新操作之后，怎样最小化对 Flash 的更新？我们现在的方案是将更新分解为插入和删除，因此需要建立一个删除列表，那么这个删除列表又应该如何维护？
- 怎样设计缓冲策略，在尽量少使用内存的情况下最小化索引维护的代价？
- 无论如何存储和维护，垃圾回收都是不可避免的。那么采用什么策略才能顺应系统的总体目标，同时又不破坏损耗平衡？

## 3.3 Flash-based DBMS 的查询处理和优化

在任何一个数据库管理系统中，查询处理和优化都是系统实现中的关键。考虑到 Flash 的特殊应用环境，查询处理所能使用的资源可能受到巨大限制；考虑到 Flash 的读写特性，查询优化的目标和内容也会有所变化；考虑新的存储和索引方案，查询处理和优化的实现策略必然要以此为基础。查询处理和优化的算法应满足以下约束：

- 不应依赖于 RAM，但又能充分利用可用的 RAM。也就是说，当 RAM 资源紧张时，系统能够正常运行，但是 RAM 资源充足时，系统又可以利用这些资源来提高效率；
- 合理利用已有的存储和索引结构。最大化发挥这些数据结构的优势，以充分利用 Flash 的特性（如读写速度不等）。

### 3.4 Flash-based DBMS 的事务和恢复

在数据库管理系统中，要保证数据的一致性和持久性，事务处理是必不可少的。故障之后的恢复通常是利用日志记录来重做或撤销事务。然而，在 Flash 上以何种方式来管理日志记录也是值得考虑的问题。可研究的问题包括：

- 首先，由于 Flash 数据不可覆盖的特性，数据本身往往就包含了故障恢复所需信息，日志的内容可以适当缩减；
- 其次，检查点的设置也不应该是随意的，而是根据存储和索引模型针对 Flash 空间利用情况作适当优化。

### 3.5 其它（安全、并发等）

在上述研究基础上，加入数据库的安全访问控制和数据加密算法，并考虑支持并发操作。尤其是在并发操作中，各种锁的实现需要充分考虑数据的存储和索引结构，利用 Flash 特性来最大化并发度。

## 四、结束语

当数据库邂逅 Flash，不变的是什么，改变的又是什么？在人类需求的加速膨胀中，基于 Flash 的数据库到底会扮演怎样的角色？对人们的生活和工作方式将会产生怎样的影响？

我们有太多的资料要搜集，我们有太多的问题要思考，我们有太多的推断要证明。

我们，一直在努力。

## 参考文献：

- [1] M-Systems white paper. Two Technologies Compared: NAND vs. NOR. July, 2003.
- [2] Samsung data sheet. 1G x 8 Bit / 2G x 8 Bit / 4G x 8 Bit NAND Flash Memory. Nov, 2005
- [3] E. Gal and S. Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2):138–163, 2005.
- [4] H. Garcia-Molina, J. Ullman, J. Widom. Database System Implementation. New Jersey: Prentice Hall, 2000.