arXiv:2603.05632v1 [cs.DB] 5 Mar 2026

面向内存受限的闪存嵌入式设备的空间高效B树实现

Nadir Ould-Khessal、Scott Fazackerley 和 Ramon Lawrence

不列颠哥伦比亚大学

摘要。 用于农业、环境和工业监测的小型数据采集设备，为物联网 (IoT) 应用提供了支持。鉴于这些设备在数据采集中的关键作用，有必要进行优化，以改进设备端数据处理。边缘设备计算使数据能够在更接近采集地点的位置进行处理，从而减少网络传输量。B树已针对服务器和固态硬盘上的闪存存储进行优化，但这些优化通常需要嵌入式设备所不具备的硬件和内存资源。本研究的贡献在于，为内存受限的嵌入式设备开发了多种B树变体，并对其进行了实验评估。实验结果表明，即使是最小型的设备也能高效地执行B树索引，而且采用针对存储介质的优化具有显著的性能优势。

关键词： B树 · 嵌入式设备 · 数据索引与存储 · 物联网

1 引言

部署在网络边缘的设备正在采集并处理大量数据。在将数据发送到服务器之前先在设备上进行数据处理，可以改善响应时间和能源效率，并减少通过网络传输的数据量。最小的嵌入式设备具有独特的硬件与性能特征，给数据处理带来挑战。这些设备的内存可能非常小，介于4到64千字节之间，中央处理器能力有限，并且使用没有文件系统的原始闪存存储。数据处理以插入操作为主，与所采集数据相关的查询数量有限。

本工作针对该环境对B树 [3]进行了优化。先前的研究已将B树适配到基于闪存的存储，例如固态硬盘（SSD）。一项关于闪存索引的综述 [9] 概述了30多种实现。一种常见方法是利用可用内存推迟写入，从而实现批处理和顺序写入，以减少代价高昂的闪存写入次数。系统要么采用日志记录方法 [16, 18, 24, 30]，要么在写入前缓冲页面更新 [4]。性能最高的算法利用了SSD中固有的并行性 [9]。

本工作的较短版本曾发表于 CoopIS 2024 [21]。本文在会议版本的基础上进行了扩展，补充了关于闪存索引技术的背景介绍，并更详细地描述了虚拟映射和恢复

机制，进一步讨论了存储管理策略，并更全面地呈现了实验评估。

在内存受限嵌入式设备上实现B树，相较于服务器上的索引处理，面临着独特的挑战。最重要的问题是内存有限。这要求算法不仅考虑缓冲区所占用的内存，还要考虑其总内存消耗。实现本身以及状态变量所使用的内存同样重要。尽管所处理的数据规模小了几个数量级，但RAM与存储比率相对更低。典型服务器的RAM容量至少相当于数据存储大小的1%，而许多部署环境拥有足够的内存，可以存储数据集的很大一部分 [2]。嵌入式设备的内存容量可能不到数据大小的1%，且绝对内存容量非常小。在这种环境中，每个字节都至关重要。

第二个因素是，闪存存储可能由原始闪存芯片组成，而不是SD卡或固态硬盘等封装产品。这会在两个重要方面影响实现。首先，实现必须负责管理设备上的物理内存，包括磨损均衡、页放置和空闲空间管理。在没有闪存转换层（FTL）的情况下[5], ，算法必须在使用最少内存的同时，将逻辑页映射到物理页。其次，并行性的机会更少。中央处理器通常是单线程的，存储设备一次只能响应一个请求，而通过总线传输数据也是瓶颈。输入输出性能受总线速度限制，而不是存储设备带宽。

本文针对内存受限的嵌入式环境，提出并评估了B树的优化方案及变体。具体贡献包括:

– 通过使用虚拟映射，降低B树更新写放大的方法。虚拟映射支持在不重建B树索引结构的情况下移动物理页。– 一种利用虚拟映射、面向原始 NAND 闪存存储的B树变体。– 一种面向支持页覆写的内存（如 NOR闪存和 DataFlash [1]）设计的B树变体。– 在两种硬件平台和三种存储器类型上进行实验评估，展示针对存储的B树优化如何利用硬件的独特特性来最大化性能。– 分析可用内存较少时，B 树索引最有效的内存使用方式。

本文的组织结构如下。第2节介绍 B树数据结构以及用于基于闪存的索引的实现背景。第3节介绍本研究的动机环境。第4节讨论 B 树实现的优化，重点介绍管理虚拟映射和页覆写的新技术。第5节给出实验结果。最后，本文总结全文并展望未来工作。

2 背景

B树 [3] 是面向数据库应用的高性能索引结构。大多数实现采用 B^+-树变体 [6]，其中数据记录仅存储在叶节点中，而内部节点则包含用于导航的键。图 1 展示了一个 B^+-树示例，其中每个节点最多包含 3 个条目。

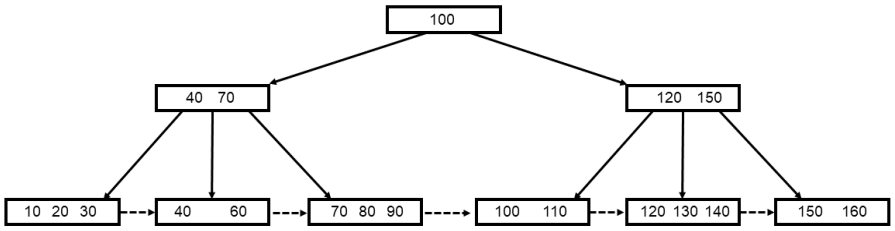


图 1：B^+树示例

B树实现有着悠久的历史，最初主要面向硬盘。Graefe [11] 详细讨论了 B 树实现面临的众多挑战、技术和最佳实践。Foster B树 [12]结合了多种 B树变体的优点，包括写优化B树 [10]，以支持更高的并发性和在闪存存储中的高效页迁移。Foster B树在 B 树结构中去除了兄弟指针（图中的虚线指针），仅依赖父子指针。

虚拟内存和影子分页技术 [25]已应用于 B 树 [13, 23]。Bw树 [17] 使用映射表将逻辑指针地址转换为物理页位置。这样，页就可以写入任意存储位置，而无需更新从叶节点到根节点整条路径上的指针。映射表的性能至关重要。

从硬盘迁移到基于闪存的存储和固态硬盘（SSD），使人们重新重视 B树以及针对闪存特性进行优化 [9]。闪存的主要优势在于，与硬盘较长的寻道时间相比，其随机访问性能显著更高。这促使人们在 SSD 上存储数据时，采用比在硬盘上更小的 B树节点。一个挑战是处理闪存的这一特性：在大多数情况下，物理页无法原位覆盖，必须先执行块擦除。写入操作通常比读取操作成本更高，导致性能不对称。

针对闪存特性优化的 B 树实现会优先考虑读取而非写入，使用内存缓冲区和日志记录来减少写入次数，并延迟随机写入，再将其批量处理为顺序写入。固态硬盘内部配备中央处理器和内存，可以同时处理多个请求。随着固态硬盘技术的发展，固态硬盘内部硬件通过闪存转换层（FTL）算法智能管理物理内存页，从而缩小了读取与写入性能之间的差距。

要理解一个根本性挑战，不妨考虑将键 50 插入图 1 中的 B 树。键 50 将被插入当前包含 40 和 60 的第二个叶节点。在硬盘上，包含该节点的页会被更新，并在磁盘上原位写回。无需修改 B 树的结构。在固态硬盘上，B 树实现也可以在逻辑上覆盖该页，但固态硬盘必须通过闪存转换层（FTL）将该页写入另一个物理位置，因为不支持在不擦除的情况下覆盖同一位置。

对于没有闪存转换层（FTL）和文件接口的原始闪存芯片，必须改变 B 树实现。无法在物理上覆盖已更新的页，而写入新的页位置则需要更新 B树指针。更新一个指针可能需要级联更新其他页。在我们的示例中，插入 50 并将更新后的页写入新位置，需要更新包含 (40,70) 的父节点中的指针，进而触发根节点的更新和写入。此外，如果使用兄弟指针，还需要更新包含 (10,20,30) 的叶节点。减少这种写放大对于确保性能达到可接受水平至关重要。

一项关于基于闪存的索引的综述 [9]概述了各种 B 树实现。写优化树等缓冲方法 [4] 会将节点缓存在内存中，并以批处理方式写入，从而摊销写入成本并避免小规模随机写入。日志记录方法 [16, 18, 24, 30] 会将更改记录在内存中或闪存数据页上，而不是立即更新。在我们插入键 50 的示例中，缓冲方法可以将叶节点及其父节点缓存在内存中，待发生多次更新后再写入这些节点。日志记录方法则会将插入 50 的操作作为一条日志记录存储在单独的内存区域中，并推迟对树的更改。日志区域写满时，再以批处理方式更新树。利用内存进行缓冲或日志记录，可以充分发挥固态硬盘硬件资源和功能的并行使用能力，从而提升性能。

原始闪存上的 B 树实现通常会修改 B 树节点结构，以减少写放大。这些实现会管理内存，包括空闲空间和垃圾回收。有些实现会利用闪存的特性，例如页部分更新或覆盖写入 [14, 15]。基于缓冲和延迟写入、面向闪存和固态硬盘的 B 树实现，如果不针对可用内存较少的情况进行调整，就无法在内存受限的嵌入式设备上运行。有一种面向使用有限内存的嵌入式设备的 B 树实现 [19] ，但它并未针对原始闪存进行优化，而且依赖基于文件的存储。

嵌入式设备的其他索引结构包括 Antelope [29], Micro-Hash [32], PBFilter [31], 和 SBITS [8]。这些索引结构通常用于顺序时间序列数据，采用顺序写入并最大限度地减少内存使用。它们不支持像 B 树那样对非顺序数据进行索引。

3 动机环境

目标环境是主要用于日志记录和监控应用的资源受限的嵌入式设备。设备的中央处理器速度可能为 16

兆赫至 128 兆赫，RAM 容量为 4 至 64 千字节，并使用 NOR闪存、NAND 或 SD卡存储。闪存存储可能没有闪存转换层或文件系统。闪存存储容量取决于数据采集需求，可能为 8 兆字节至 1 吉字节。这些硬件规格（见图2）与 1970 年用于早期 B树实验的 IBM 360 机器相当。

设备	CPU	RAM 输入/输出速率	
M0+SAMD21	48 兆赫	32 千字节	450 千字节/秒
PIC24FJ1024	16 兆赫	32 千字节	200 千字节/秒
IBM 360/44 [3]	1.6 兆赫	32-256 千字节	156 千字节/秒

图 2：1970 年的嵌入式设备和原始实验性 B 树硬件

页是闪存存储的 I/O 单位，大小介于 256 至 4096 字节之间。页在写入前必须先擦除。擦除以块为单位进行，一个块由连续的页组成。由于擦除块的成本较高，写入不会在原位置进行。如果使用 SD卡存储，闪存转换层（FTL）会写入新的物理位置，并更新其内部的逻辑页到物理页映射。使用不带 FTL 的原始存储器时，算法必须管理物理页分配、垃圾回收和磨损均衡。某些 NOR 闪存支持有限的页覆写 [1]。只要没有任何位从 0 变为 1，就可以覆写页。位从 1 变为 0 是允许的。实现方法是先将页初始化为全 1，然后向页中尚未写入的空闲空间追加数据。

日志记录环境领域的特点是数据访问要求非常严格且具体。设备通常会采集时间序列数据，其中包括时间戳和传感值，后者通常是一个较小的数字。高效的插入操作是首要考量。记录插入通常按照记录时间戳的顺序以只追加的方式进行。更新和删除极少发生。当设备清除历史数据时，可能会定期进行批量删除。例如，较早的数据在不再有用或已传输到服务器进行处理后，可能会每天从系统中删除。设备上的查询处理用于按时间戳或数值进行搜索，以及执行跨时间窗口的聚合查询。这些查询由设备作为其功能的一部分生成，通常在部署前已确定。RAM 容量有限给数据处理带来挑战，因此最大限度地减少内存使用至关重要。RAM 与存储的比率低于服务器系统 [2]。例如，如果有 4 KB RAM，那么它占 400 KB 数据量的 1%，占 4 MB 数据量的 0.1%。

B 树实现应适应这些不同的设备属性。嵌入式索引有两种不同的应用场景。对仅追加的已排序时间序列数据建立索引，对于基于时间戳进行快速检索至关重要。许多以顺序写入为基础的既有索引结构（包括简单的已排序文件）都表现良好，且内存使用极少 [8, 9,

29, 31, 32]。即使针对顺序数据进行了优化，B树的性能也与这些专用索引结构相当，但并不更优 [20]。

当前嵌入式索引结构在按时间序列中的数据值进行检索时，效率并不高。数据值没有顺序，索引必须能够处理对任意位置的随机写入。此外，数据集中也可能存在大量重复数据。再者，如果传感器采集的是环境的物理测量值，数据可能具有更强的周期性，并遵循由测量值决定的函数关系。例如，温度值会随时间缓慢变化，且变化方式相对可预测。本研究着重于如何高效地为未排序数据建立索引。

4 B树优化

本节介绍适用于嵌入式系统的 B 树优化和实现变体。此前关于日志记录和写优化树的研究 [4, 16, 18,24, 30] 将与一种采用虚拟映射、在子节点指针发生变化时延迟写入节点的方法相结合，并进行比较。尽管先前的 B 树实现可能无法直接在此环境中运行，因为它们要么消耗过多内存，要么依赖此处不可用的操作系统功能，但其中的关键优化仍可迁移并加以调整。

4.1 缓冲与日志记录

许多针对闪存优化的 B树方案都采用缓冲更新以延迟写入，或记录变更以便之后批处理这些操作。要将这些技术应用于嵌入式环境，必须考虑到可用内存有限。通常，内存只够缓冲根节点和工作路径上的少数几个节点，不足以缓冲根节点子节点中哪怕很小一部分。直接在缓冲区中更新 B树节点并延迟写入存储的方案，性能提升有限，因为页在写入存储之前不会在缓冲区中停留足够长的时间，无法累积多次更新。

将操作记录下来以便批处理，比分别存储修改后的页更加节省内存。这样一来，有限的内存中便能容纳更多修改。以往基于日志的方法各不相同：日志可以关联整棵树、单个节点，或存储块。最节省内存的做法是为整棵树使用一个日志，使其充当写缓冲区。写缓冲区满后，对操作进行排序，并批量执行插入操作。通过一次向页中插入多条记录，这种方法减少了输入/输出操作；对于聚簇数据，性能也很好。

执行批量插入的写缓冲区优化可与其他任何 B树优化结合使用。对于对数据丢失有限制的应用，可以调整写缓冲区的大小，使其在经过一定时间或累积一定数量的记录后执行操作。

4.2 虚拟映射

固态硬盘和 SD卡等存储设备支持逻辑页覆盖写入，因为闪存转换层会将文件中的逻辑页写入转换为设备上的物理内存位置。在硬件层面，存储设备必须将逻辑页请求转换为物理位置，并维护不再有效的页的信息。对于应用而言，这一过程是透明的：设备会擦除包含无效页的块，并维护其存储分配。由于无法在原始闪存存储上原位更新，更新节点会导致数据写入新的物理页，因此必须更新指向先前物理页的 B树指针。指针更新会级联至树中的多个节点，而写入叶节点可能导致从叶节点一直到根节点的整条路径都被写入。

虚拟映射解决了在没有闪存转换层的闪存存储上写入B树节点时产生的写放大问题。由于设备内存有限，无法实现闪存转换层，因此虚拟映射是一种专用于B树的转换机制。虚拟映射由一对页面标识符组成，分别是旧页面标识符和新页面标识符 旧页面标识符 $\rightarrow$ 新页面标识符。每当将页写入新位置时，就会在映射表中插入一条虚拟映射。页面标识符通过起始内存地址加上页面标识符乘以页大小所得的偏移量，映射到物理地址。例如，插入键 50 时，包含 (40, 60) 的节点会更新并写入新位置。包含 (40, 70) 的父节点需要更新指针，以指向新位置。此时不修改父节点，而是创建一条虚拟映射。每当遍历该指针时，都会使用映射表查找是否存在新位置。

为尽量减少映射数量，与Foster B树 [12],类似，不设置兄弟指针。仅有父指针时，指向任一给定节点的指针至多一个。虚拟映射类似于Bw树所使用的映射表 [17] ，但因内存有限，映射表大小有界，且表中仅包含所需映射。给定B树节点中作为指针的页面标识符，若表中未找到映射，则该页面标识符即提供该节点当前的物理位置。因此，这些映射更类似于物理重定向，而非逻辑到物理的映射。一条映射不需要额外的输入/输出操作，也不会被多次链接。考虑包含(40, 60)的节点被多次写入不同位置 $L_1, L_2, ..., L_N$，而父节点(40, 70)的指针存储指针值 L_1。写入位置 L_2时，映射 $L_1 \rightarrow L_2$ 被加入表中。此后每次写入仅更新映射 $L_1 \rightarrow L_3$、 $L_1 \rightarrow L_4$、...，直至最终$L_1 \rightarrow L_N$。无需保存任何中间映射，因为只有一个节点使用该映射，且其中包含值 L_1。在执行任何输入/输出之前，会先搜索映射表，以判断页的物理位置是否已改变。在所有情况下，检索该页都只需一次输入/输出。

虚拟映射必须驻留在内存中。否则，每次指针查找都需要执行一次输入/输出，以确定映射是否存在。虚拟映射记录的大小为 4 或 8 字节，具体取决于存储页的数量。映射表的大小可配置，但通常只有约 1 到 2 千字节。我们的实现

经过调优，在支持快速查找的同时尽量减少内存占用，并使用带双重哈希的内存哈希表来处理冲突。可以在插入操作期间或 B 树操作之间从映射表中移除映射。在插入操作期间，可以确定通往叶节点的路径上每个节点的映射数量；如果映射数量超过一定值，就将该节点写入存储。只有在写入的节点包含 2 个或更多映射时，映射总数才会净减少。这是因为每当更新节点时，都可以移除其映射（因为其指针已更新），但其父节点需要为该节点的新位置创建一个映射。任何时候都可以从根节点开始遍历，找出可移除映射的节点。将从叶节点到根节点的整条路径写入存储，可以移除该路径上的所有映射。如果没有空间存储映射，就将需要该映射的节点写入存储。有足够空间存储所有映射时，不会产生写放大。使用虚拟映射可以将页顺序写入存储。当页更新时，会将其写入下一个连续的页位置，并创建一个虚拟映射。这对某些设备有性能优势，也简化了空闲空间管理和垃圾回收。

4.3 页覆写

对于原始内存存储，应用程序负责管理物理内存，包括空闲空间管理和块擦除。通常，已写入的页不能原地覆写。不过，有些存储设备支持有限的页面覆写，可以利用这一特性来提升性能 [1]。具体而言，某些 NAND 闪存支持部分页写入，而 NOR 闪存和 DataFlash 产品则支持在满足特定条件时覆写整页。由于 NOR 闪存的特性，整页都可以覆写，但仅支持将位从 1 改为 0 [1, 7]。实际上，这使得可以写入页中之前尚未写入的部分。

覆写可以提升写入性能并减少擦除次数。在 [14],中，B树节点经过修改后会存储字节级差异。我们的实现按线性插入顺序（即不排序）存储 B树节点，这种方式适合覆写。每条记录都会追加到页中。每条记录关联两个位：计数位和有效位。两个位向量都初始化为 1。将记录插入页中时，计数位从 1 置为 0，有效位则保持为 1。如果移动或删除记录，则将其有效位置为 0。通过累加计数位为 0 且有效位为 1 的所有记录，确定页中的有效记录数。这种结构使所有更新都只需执行存储器支持的从 1 到 0 的写入。由于键不再排序，因此采用线性搜索。

4.4 空闲空间管理

对于不提供这些功能的存储，实现了空闲空间管理和垃圾回收。存储被视为一个循环数组。任何时候，当前写入位置之前都有若干块已被擦除。在

擦除一个块之前，会确定有效页，并将其缓冲到内存或存储中。块擦除后，有效页会被写回该块中的原始位置，因此无需更新树中的任何映射。

判断一个页是否有效有两种方法。第一种方法是为每个页关联一个空闲空间位。页不再使用后，将该位设为可回收。如果内存不足以存储位向量，则可以采用页面搜索策略来判断该页是否有效。判断存储中的任意给定页是否为有效页的算法是：从存储中读取该页P，从页中选取一个键 K，然后在当前树中搜索 K。如果未找到 K，或者遇到不同的页，那么 P 就不是有效页，可以将其删除。

4.5 恢复

每个页面头部都存储一个页面标识符和一个前一页面标识符。首次写入的新页，其前一页面标识符为 -1。写入现有页（页面标识符，前一页面标识符）的新版本时，如果存在从前一页面标识符 →页面标识符的映射，则新页的前一页面标识符为前一页面标识符；否则为页面标识符。这样便可以追踪页的来源，并在设备故障或重启时恢复虚拟映射表。

重启时，存储会从最后一次写入的页开始向前扫描。在扫描过程中，通过查找页头中设置了根标志且页面标识符最大的页，确定最新的根节点。读取带有页头（页面标识符、前一页面标识符）的页时，在映射表中查找 前一页面标识符。如果未找到映射，则以 前一页面标识符 $\rightarrow$ 页面标识符的形式将映射添加到映射表中。如果找到映射 前一页面标识符 $\rightarrow$ 页面标识符2，且 页面标识符2 >页面标识符，则 页面标识符页是较旧版本，并在空闲空间位图中标记为空闲页。

考虑位于位置 5 的叶节点，由于插入三个键而被写入三次。第一页的页头包含（页面标识符、前一页面标识符），其值为（5、 -1）。第二页的页头为（6、 5）。第三页的页头为（7、5），第四页的页头为（8、5）。从最后一页开始向前扫描存储，会添加映射 $5 \rightarrow 8$，并将第 5、6 和 7 页标记为空闲空间。

如果插入操作期间发生故障，由于先前的页数据仍保存在存储中，因此可以恢复到一致状态。如果故障发生在叶节点写入存储之后，且无需修改其上方的内部节点，则只需恢复虚拟映射表即可达到一致状态。如果叶节点已在存储中更新，但内部节点的任何分裂尚未写入存储，则通过重新执行该键的插入操作来恢复。通过与先前的节点版本进行比较，可以确定被插入的键。

4.6 实现变体

B树优化以多种实现变体的形式部署。基础B树实现执行就地更新写入。此实现仅适用于支持文件系统接口的存储设备

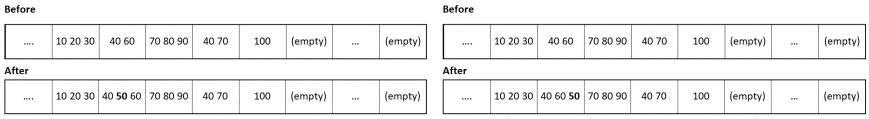
（具有底层闪存转换层）或支持按页擦除后写入的原始闪存（例如 DataFlash [1]）。虚拟映射树 VMTree 使用虚拟映射表来最大限度地减少写放大，并对存储执行顺序写入。当逻辑页被覆写时，更新后的页面会写入下一个连续的页位置，并在映射表中插入一条映射，以避免修改父页面。VMTree-OW 实现会在支持该功能的存储器上执行物理页覆盖写入。无需虚拟映射表。

所有实现都使用一个通用页缓冲区，通过最近最少使用算法缓冲常用页。根页始终缓冲在其中。每种实现还可以使用写缓冲区，以便批处理操作。

图 3 展示了不同变体如何将键 50 插入图 1 所示的树中。B树不管理空闲空间或存储器，因为它依赖存储设备来处理这些事务。B树对逻辑页执行原位写入，而物理位置可由闪存转换层决定，位于存储器的任意位置。该图展示了将键 50 添加到现有逻辑页中。B树无法控制物理页位置。

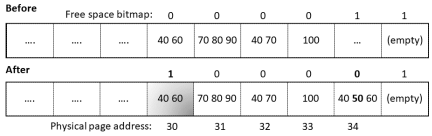
VMTree-OW 对物理页执行覆盖写入。键 50 被插入到下一个位置的现有物理页中。该位置的计数位被设为 0，表示该位置存在有效记录，而有效位仍为 1。这种覆盖写入可确保页上的所有位都从 1 变为 0。

VMTree 管理其空闲空间和物理存储。由于 VMTree 按顺序写入且不会覆盖任何页，因此它在存储地址空间中的推进速度会快于另外两种算法。空闲空间位图用于判断某个页是否已被使用（0）或可用（1）。添加键 50 后，会生成一个包含（40、50、60）的新页，并将其存储在存储设备当前的写入位置，即物理页地址 34。地址 30 处前一页的位向量被设为 1，表示该页可以重新使用。映射表中新增一项映射 $30 \rightarrow 34$，因此无需更新地址 32 处的父节点。



(a) B树

(b) VMTree-OW



(c) VMTree

图3：插入键50

5 实验结果

在多个数据集上，研究人员在若干硬件平台上对 B 树实现进行了比较。硬件平台和存储设备见表4。数据集的特征见表5。结果为 3 次运行的平均值。

设备	存储页大小	R（千字节/秒）	W（千字节/秒）	读写比
32位 SAMD21 SD卡	512 字节	500 ; 400	500 ; 215	1 ; 1.9
32位SAMD21DataFlash	512字节	475	35	13.5
16位PIC	NAND	2048字节	203	187	1.1
16位PIC	SD卡	2048字节	198 ; 194	108 ; 93	1.8 ; 2

图 4：示例嵌入式设备与内存配置

数据集	密钥大小	记录大小	描述
随机	4 字节	16 字节	随机 32 位整数
环境	8 字节	8 字节	来自 [32] 的时间序列数据。数据值索引（温度、气压或风速）。
健康	8 字节	8 字节	WESAD 时间序列数据 [26] 监测患者的心率、温度、运动患者心率、温度、运动。

图 5：实验数据集

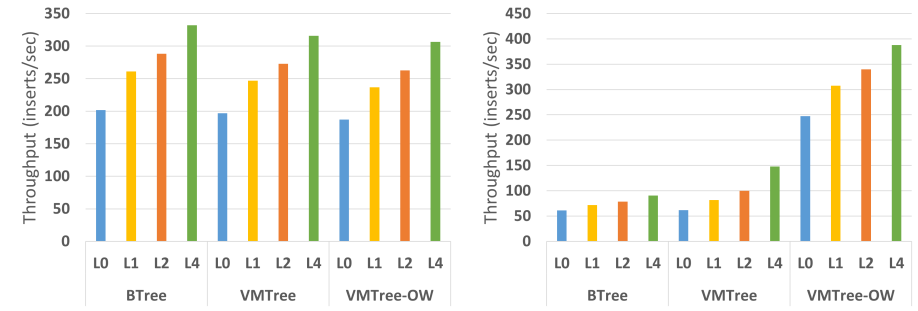
图 4 展示了每种设备的读写性能，其中第一个数值表示顺序性能，第二个数值表示随机性能。SD 卡的随机性能低于顺序性能，且不同卡之间存在差异。原始 NAND 和 DataFlash 芯片在顺序和随机输入/输出操作中的性能相同。DataFlash 芯片支持页级写入时同时擦除，而无需擦除整个页块。这种操作的开销高于写入已擦除的页。35 千字节/秒的写入带宽对应先擦除再写入操作，这是执行原地写入的 B 树实现所必需的。实验开始前会预先擦除存储设备。

分配给各实现的缓冲内存为 $M=3$ 个页缓冲区。VMTree 算法还需要额外的内存来存储映射表。32 位 SAMD21 平台使用的页大小为 512 字节，而 16 位平台的页大小为 2048 字节。在 SAMD21 平台上，消耗的总内存为 3141 字节，包括 3 个 512 字节的页、330 字节的状态变量、251 字节的位向量以及 1024 字节的映射表。在 PIC 平台上，消耗的总内存为 8866 字节，包括 3 个 2048 字节的缓冲页、547 字节的状态变量、127 字节的位向量以及 2048 字节的映射表。B树和 VMTree-OW 不需要映射表或空闲空间位向量所占用的内存。使用写缓冲区时，还可以选择消耗额外内存。

5.1 随机数据

随机数据集包含 10,000 条记录，每条记录为 16 字节，并带有一个 4 字节的随机整数键。实验插入了 10,000 条记录，随后执行了 10,000 次查询。在 SD卡上，VMTree 的运行时间和输入输出性能与 B 树实现相差不超过 3%。由于 SD卡自带闪存转换层，因此维护虚拟映射表并无优势。VMTree 执行顺序输入/输出，而 B树 执行随机输入/输出；不过，这一优势在测试结果中并不明显，因为 SD 卡上的随机输入/输出仅慢约两倍。在不支持硬件级覆盖写的 DataFlash 上，VMTree-OW 慢了 8%，因为其无序叶节点结构需要更多空间和时间来处理。在 DataFlash 上，VMTree 比 B树慢约 2%。这种内存支持硬件级覆盖写，因此 VMTree-OW 凭借适合覆盖写的页结构，性能提升了四倍。

图 6 显示了不同写缓冲区大小下各 B树变体的插入吞吐量（例如，*L1*表示一个写缓冲区）。引入写缓冲区可提升所有实现的性能。写缓冲区带来的益处与批量索引插入相同 [22]：显著减少输入/输出操作和总耗时。更大的写缓冲区增加了对同一叶节点的写入进行批处理的机会，从而进一步减少输入/输出操作和耗时。VMTree 的收益略高，这是因为批处理操作减少了树遍历期间的虚拟映射查找次数。对于查询，所有实现的读取输入/输出操作数都相同。VMTree 和 B树的查询时间相差不超过 2%。VMTree-OW 慢 10% 至 20%，因为其节点未排序，需要进行线性搜索。



（a）SD卡　　（b）DataFlash

图 6：32 位 ARM 插入吞吐量

在较小的16位PIC设备上，结果也类似。图7展示了B树和VMTree在SD卡和NAND上的随机插入相对性能。由于NAND内存不支持硬件覆盖写入，因此未测试VMTree-OW；B树也无法在NAND内存上运行，因为NAND不支持对单个页进行原位写入。

对于图中的成对比较，百分比差异按 $((tree1 - tree2)/tree2)$ 计算。对于VMTreeSD与BTreeSD的比较，这意味着VMTreeSD的读取操作多2.2%，写入操作多0.5%，耗时长9.1%。

由于虚拟映射和空间管理带来了一些额外的输入/输出操作及开销，SD卡上的VMTree比B树稍慢。NAND上的VMTree性能优于B树和运行在SD卡上的VMTree。它执行的输入/输出操作次数相同，但运行在写入带宽更高的芯片上。这些结果表明，VMTree实现可以在原始存储器上以极高的效率运行。这是一项显著优势，因为NAND芯片的成本为2至4美元，比SD卡（10至20美元）便宜5倍，而且更易于部署在小型设备上，因为它可直接集成在电路板上。这些结果证明了针对特定存储属性定制B树实现的优势。加入写缓冲区可大幅提升性能，同时无需占用大量内存。

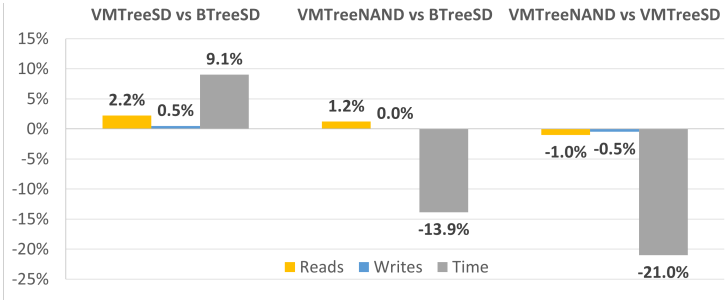


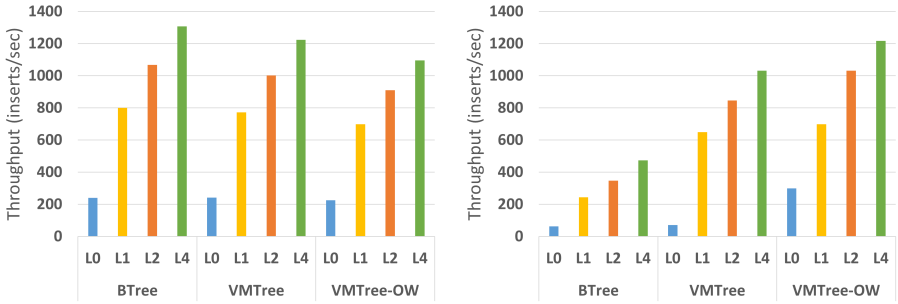
图7：16位PIC在SD卡和NAND上的随机插入性能

5.2 传感器数据

评估了两个传感器数据集：环境气象站数据 [8, 32]，包含温度、降水量、湿度和风速的每小时采样值；以及 WESAD（可穿戴设备压力与情感检测）健康数据集 [26]，包含传感器测量值，包括心电图（ECG）、呼吸、体温和三轴加速度。B树索引存储 8 字节的索引条目，每个条目由数据值（即温度或 ECG）和用于在时间序列数据集中进行检索的记录标识符组成。尽管数据按时间戳排序，但并未按数据值排序。数据值可能存在大量重复，并具有时间聚簇特征。B树是一个辅助索引，支持按数据值搜索的查询。分配的缓冲内存为 M=3 个页缓冲区。实验插入了 10,000 条记录，然后在 B树上运行了 10,000 个查询。

32 位 ARM处理器上，SD卡存储和 DataFlash 的插入吞吐量结果如图 8 所示。在 SD卡上，各实现的相对性能与随机数据集相同，VMTree 略

慢于 B树。在 DataFlash 上，VMTree-OW 的性能比 B树快四倍。VMTree 比 B树快，因为它在预擦除存储区域上采用顺序写入模式，比 B树的原地擦除后再写入更高效。由于聚簇，写缓冲区相较于随机数据带来的收益更大。温度数据变化缓慢，使用一个写缓冲区对插入操作进行批处理，可使所有 B树变体的输入/输出和耗时减少 63-72%。



(a) SD卡 (b) DataFlash

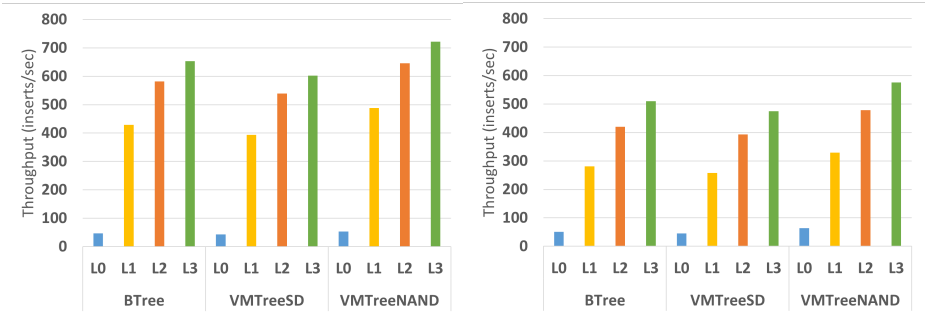
图8：温度数据的插入吞吐量（ARM）

健康数据以及其他环境变量（包括湿度、气压和风速）呈现出相似的性能特征。尽管输入/输出操作节省比例会因重复数据量和时间聚类情况而异，但所有传感器数据在使用写缓冲区时都显著减少了输入/输出操作。无论数据集和写缓冲区大小如何，B树变体的相对性能都保持一致。在所有数据集和写缓冲区大小下，查询性能均无差异，因为输入/输出操作次数完全相同。

在16位PIC平台上，结果与32位ARM平台一致。图9展示了SD卡上温度数据集和健康数据集的插入吞吐量。增加一个写缓冲区后，温度数据的吞吐量提高了9倍，健康数据则提高了5倍。VMTree在NAND上的性能始终最高。

5.3 内存使用优化

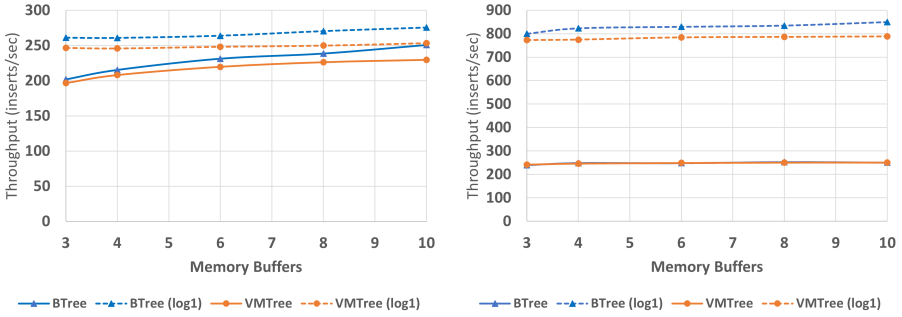
一项关键优化是确定如何恰当地使用额外的 RAM。对于所有 B树变体，支持读取、写入和树维护所需的缓冲区数量至少为三个。分配额外内存有两种方式：通用页缓冲区和写缓冲区。通用页缓冲区使用 LRU缓冲区将页存储在 RAM 中。写缓冲区会先缓存操作，再对树执行这些操作；额外的页则可以在修改树之前批处理更多操作。实验使用了随机数据集和温度数据集，将页缓冲区数量从3个调整到10个，并比较了使用0页或1页写缓冲区（log1）的情况。实验测量了写入和读取吞吐量。



(a) 温度数据 (b) 健康数据

图9：16位PIC插入吞吐量

图10和图11展示了写入和读取吞吐量的结果。



(a) 随机数据 (b) 温度数据

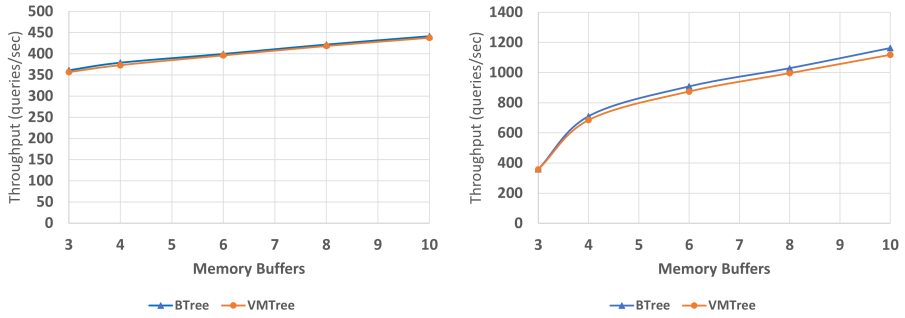
图10：写吞吐量随内存变化

就写吞吐量而言，在没有写缓冲区的情况下增加页缓冲区，可使B树对随机数据的吞吐量提高5%至25%，对温度数据的提高不到5%。温度数据的缓冲区命中率相当高，增加更多缓冲区也不会显著提高。VMTree与之类似，只是由于缓冲区命中率较低，随机数据的性能提升较小，介于3%至14%之间。增加更多缓冲区不会改变写入I/O，但会因更多缓冲区命中而减少读取操作。添加写缓冲区的影响要大得多，由于聚簇，温度数据会显著增加。

写缓冲区不会影响读取吞吐量。增加页缓冲区可以提高缓冲区命中率，从而改善读取性能。对于重复率高且具有聚簇特性的数据（如温度数据），这种改善尤为明显。从

将页缓冲区从三个增加到四个，可使系统在内存中缓冲从根节点到叶节点的完整工作路径，从而在执行分裂和树维护时，无需重新读取通往叶节点路径上的内部节点。

如果可以利用写缓冲区延迟并进行批量更新，那么相比于为通用页缓冲区分配内存，将更多内存专用于写缓冲区会带来更大的收益。拥有足够的页缓冲区来缓冲一条根到叶路径是非常有价值的。



（a）随机数据　　（b）温度数据

图 11：读取吞吐量随内存变化　　y

5.4 改变数据大小

VMTree需要比B树和VMTree-OW更多的内存来维护其虚拟映射表和空闲空间位图。当VMTree在存储空间中绕回时，还必须通过移动有效页来执行垃圾回收，然后再擦除块。为量化这些影响，我们对100,000条随机数据和温度数据记录进行了测试。在SD卡上进行测试，便于与无法在原始NAND上运行的B树进行比较。存储空间设为5000页，M=4 用于页缓冲区，映射表大小为4096字节。对于100,000条记录，VMTree执行了超过135,000次写入，需要遍历存储空间27次。

对于随机数据，记录数为 10,000 时，VMTree 的写吞吐量与 B树相差不到 1%。记录数达到 100,000 时，B 树的吞吐量则高出 22%。造成这一差异的原因是映射表接近满载，导致输入/输出操作增多。对于温度数据集，映射表仅占满 20%，两种算法在耗时和输入/输出操作方面几乎没有差异。相比存储的垃圾回收，映射表中的冲突所导致的额外输入/输出操作对性能的影响要大得多。读取性能没有显著差异。两种变体都能扩展到更大的数据规模。

5.5 不同的 SD 卡硬件

不同制造商生产的 SD卡，性能各不相同。VMTree 与 B树之间的性能差异会受到顺序写入与随机写入吞吐量之比的影响。VMTree 执行顺序写入，其性能可能优于 B树执行的随机写入。在顺序写入与随机写入吞吐量之比为 2 的 16 吉字节 SD卡上进行的实验，并未显示出显著差异。我们还测试了五张来自不同制造商（Lexar、Sandisk、Kingston）的 SD卡。顺序写入与随机写入吞吐量之比介于 1.5 倍至 4 倍之间。VMTree 相较于 B树的性能影响不到 5%。尽管较旧的 SD 卡可能具有更高的顺序写入与随机写入吞吐量之比，但测试的卡并未表现出太大差异。这可能是由于中央处理器和总线速度较慢所致：所有卡的顺序写入性能都受总线速度限制，而非 SD卡的写入速度。

5.6 讨论

实验结果展示了不同B树变体的性能差异，以及针对存储特定属性进行适配的优势。总体而言，对于无需与原始物理存储交互的基于文件的存储，基础B树实现是最佳选择。VMTree 在某些数据规模下性能具有竞争力，但除非存储的随机写入性能与顺序写入性能存在显著差异，否则管理存储所带来的额外开销并无益处。

VMTree 在原始 NAND 闪存上运行时性能出色，且仅使用 3 到 4 千字节的内存。这使得B树能够应用于以往无法使用B树的小型嵌入式设备。对于支持页覆写的设备，VMTree-OW 可显著提升性能，而且与 VMTree 不同，它不需要虚拟映射表。写缓冲区支持批量插入，能显著提升性能，尤其适用于传感器数据。如果应用场景能够容忍缓冲数据丢失，那么在可用内存较多时，添加写缓冲区的效果最为显著。

6 结论

小型嵌入式设备是物联网应用中的关键数据采集器。提升这些内存受限设备上的数据处理与索引性能，可以降低能源使用和网络传输。本研究开发、实现并实验评估了若干经过优化的B树变体。虚拟映射方法可减少写放大，并允许在无需文件系统接口的情况下，于原始闪存上实现高效的B 树实现。第二项关键优化是应用缓冲操作，其做法与先前面向服务器的B树研究相同。在应用场景允许的情况下，写缓冲对

许多传感器数据集具有显著收益，插入带宽可提高3至5倍。针对支持页覆写的内存进行优化，也可使速度提升4倍。总体而言，这三种不同的优化实现可部署于广泛的设备和存储器类型，并支持高效的B树索引，同时仅需约4千字节内存。

未来的工作将继续针对特定存储器类型和设备改进优化，并研究采用压缩等其他 B树性能优化是否有益于传感器数据采集环境。该索引将集成到 EmbedDB [27, 28] 中，为嵌入式系统提供关系型数据库和键值数据库。VMTree 的源代码位于 https://github.com/ubco-db/vmtree。

参考文献

1. Adesto Technologies：AT45DB641E 64-Mbit DataFlash。https://digikey.com/en/htmldatasheets/production/1376615/0/0/1/at45db641e-uun2b-t（2024年） 2. Appuswamy, R., Graefe, G., Borovica-Gajic, R., Ailamaki, A.：三十年后的五分钟法则及其对存储层次结构的影响。《ACM通讯》**62**3318163 3 11 114 120 2019 10 1145 ( )，– ( )。https://doi.org/ . / . Bayer, R., McCreight, E.M.：大型有序索引的组织与维护。《信息学报》**1**，173 00288683 4 189 1972 10 1007 – ( )。https://doi.org/ . /BF . Bender, M.A., Farach-Colton, M., Jannen, W., Johnson, R., Kuszmaul, B.C., Porter, D.E., Yuan, J., Zhan, Y.：Bε树与写优化简介。《Usenix杂志》**40**(5)（2015年）5. Chung, T.S., Park, D.J., Park, S., Lee, D.H., Lee, S.W., Song, H.J.：闪存转换层综述。《系统架构杂志》**55**(5), 332 – 343（2009年）。https://doi.org/10 005 6 1016 2009 03 . /j.sysarc. . . . Comer, D.：无处不在的B树。《ACM计算调查》**11**(2)，121–137（1979年6月 356776 7 10 1145 356770）。https://doi.org/ . / . . Fazackerley, S., Penson, W., Lawrence, R.：串行NOR DataFlash存储器的写入改进策略。载于：2016年IEEE加拿大电气与计算机工程会议（CCECE）。第1–6页（2016年）。https://doi.org/10 7726758 8 1109 2016 . /CCECE. . . Fazackerley, S., Ould-Khessal, N., Lawrence, R.：内存受限嵌入式传感器设备上时间序列数据的高效闪存索引。载于：第10届国际传感器网络会议论文集，SENSORNETS 2021。第92 0010318800920099 9 99页。SCITEPRESS（2021年）。https://doi.org/ . / . Fevgas, A., Akritidis, L., Bozanis, P., Manolopoulos, Y.：闪存存储设备中的索引：挑战、当前方法与未来趋势综述。《VLDB期刊》**29**8 10 1 273 311 2020 10 1007 00778 019 00559（），–（）。https://doi.org/ . /s - - - . Graefe, G.：写优化B树。载于：第三十届超大型数据库国际会议（VLDB 2004）论文集。第672–683页（2004年）。https://doi.org/10 7 11 1016 978 012088469 8 50060 . /B - - . - . Graefe, G.：B树日志记录与恢复技术综述。《ACM数据库系统汇刊》**37**(1)，1 2109197 12 1 1 35 2012 10 1145 2109196：–：（）。https://doi.org/ . / . . Graefe, G., Kimura, H., Kuno, H.A.：Foster B 树。《 ACM 数据库系统汇刊》 **37**(3), 17:1–17:29（2012年 2338630 13 10 1145 2338626）。https://doi.org/ . / . . Gray, J., Reuter, A.：事务处理：概念与技术。Morgan Kaufmann（1993年）

14. Hardock, S., Koch, A., Vinçon, T., Petrov, I.：IPA-IDX：B树索引的原位追加。载于：第15届国际数据管理研讨会。第18:1–18:3页。ACM（2019年 3329929 15 10 1145 3329785）。https://doi.org/ . / . . Kaiser, J., Margaglia, F., Brinkmann, A.：通过页面覆写延长数据库应用中固态硬盘的使用寿命。载于：第6届国际系统与存储会议。第11 2485747 16 1 11 12页：–：。ACM（2013年）。https://doi.org/ . / . . Kim, B., Lee, D.：LSB树：面向NAND闪存SSD的日志结构B树索引结构。《设计自动化与嵌入式系统》**19**(1-2)，77–100（2015年）。https://doi.org/10 4 17 1007 10617 014 9139 . /s - - - . Levandoski, J.J., Lomet, D.B., Sengupta, S.：Bw树：适用于新型硬件平台的B树。载于：2013年IEEE第29届国际数据工程会议（ICDE）。第302–313页。IEEE（2013年）18. Na, G., Lee, S., Moon, B.：B树索引的动态页内日志记录。IEEE《知识与数据工程汇刊》**2**432 19 7 1231 1243 2012 10 1109 2011（），–（）。https://doi.org/ . /TKDE. . . Ould-Khessal, N., Fazackerley, S., Lawrence, R.：面向内存受限嵌入式系统的高效B树实现。载于：第19届嵌入式系统、信息物理系统及应用国际会议（ESCS'21）（2021年）20. Ould-Khessal, N., Fazackerley, S., Lawrence, R.：使用位图、分区和树评估嵌入式时间序列索引的性能。载于：SENSORNETS 2021特邀论文。传感器网络，第1674卷，第125–151页（2022年）。https://doi.org/ 10 1007 978 3 031 17718 7_7 . / - - - - . Ould-Khessal, N., Fazackerley, S., Lawrence, R.：针对内存受限闪存嵌入式设备优化B树。载于：国际协作信息系统会议（CoopIS）论文集。第327–337页。《计算机科学讲义》，施普林格（2024年）22. Pollari-Malmi, K., Soisalon-Soininen, E., Ylonen, T.：批量更新B树中的并发控制。IEEE《知识与数据工程汇刊》**8**(6)，975–984（1996年）。https://doi.org/10 553166 23 1109 69. / . . Rodeh, O.：B树、影子复制与克隆。《ACM存储汇刊》**3**(4)（2008年2月）。https://doi.org/10 1326544 24 1145 1326542 . / . . Roh, H., Kim, S., Lee, D., Park, S.：AS B 树：面向 SSD 的高效 B+ 树研究。《信息科学与工程杂志》**30**(1)，85–106（2014年）25. Rosenberg, J., Henskens, F., Brown, F., Morrison, R., Munro, D.：基于大型虚拟内存的持久化存储的稳定性。载于：安全与持久性：支持信息安全与持久性的计算机架构研讨会。第229–245页。施普林格（1990年）26. Schmidt, P., Reiss, A., Duerichen, R., Marberger, C., Van Laerhoven, K.：介绍WESAD：用于可穿戴设备压力与情感检测的多模态数据集。载于：第20届ACM国际多模态交互会议。第400–408页。ICMI '18，3242985 27 2018 10 1145 3242969 ACM（）。https://doi.org/ . / . . Schoenit, J., Akins, S., Lawrence, R.：EmbedDB：适用于小到无法运行SQLite的资源受限嵌入式系统的高性能数据库。载于：第39届ACM/SIGAPP应用计算研讨会论文集。第345–346页。SAC '24，3636116 28 2024 10 1145 3605098 ACM（）。https://doi.org/ . / . . Schoenit, J., Akins, S., Lawrence, R.：EmbedDB：面向嵌入式系统的高性能时间序列数据库。载于：第26届企业信息系统国际会议论文集——第1卷：ICEIS。第240–249页（2024年）。https://doi.org/10 0012558100003690 29 5220. / . Tsiftes, N., Dunkels, A.：每个传感器中都有一个数据库。载于：第9届ACM嵌入式网络传感器系统会议论文集。第316–332页。SenSys '11，ACM（2011年）。https://doi.org/10.1145/2070942.2070974

20 纳迪尔·Ould-Khessal、Scott Fazackerley 和 Ramon Lawrence

30. 吴，C.，郭，T.，张，L.：《面向闪存存储系统的高效 B树层实现》。ACM Trans. Embed. Comput. Syst. **6**(3), 19 (2007). https://doi.org/10 1275991 31 1145 1275986 . / . . 尹，S.，Pucheral，P.：PBFilter：一种面向嵌入式系统的基于闪存的索引方案。Information Systems **37**(7), 634 – 653 (2012年). https://doi.org/10 002 32 1016 2012 02 . /j.is. . . . Zeinalipour-Yazti，D.，林，S.，Kalogeraki，V.，Gunopulos，D.，Najjar，W.：Micro-Hash：面向基于闪存的传感器设备的高效索引结构。载于：FAST '05 文件与存储技术会议。第 31–43 页。USENIX（2005 年 12 月 13 日）