计算机科学
COMPUTER SCIENCE

GDLIN:一种利用梯度下降的学习索引

陈珊珊, 高隽, 马振禹

引用本文

陈珊珊, 高隽, 马振禹.GDLIN:一种利用梯度下降的学习索引[J]. 计算机科学, 2023, 50(6A): 220600256-6.

CHEN Shanshan, GAO Jun, MA Zhenyu. GDLIN:A Learned Index By Gradient Descent[J]. Computer Science, 2023, 50(6A): 220600256-6.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220600256

GDLIN:一种利用梯度下降的学习索引

陈珊珊　高　隽　马振禹
南京邮电大学计算机学院　南京 210003

摘　要　在大数据时代,数据访问速度是衡量大规模存储系统性能的一个重要指标,而索引是用于提升数据库系统中数据存取性能的主要技术之一。近几年,使用机器学习模型代替B+树等传统索引,拟合数据分布规律,将数据的间接查找优化为函数直接计算的学习索引(Learned Index,LI)被提出,LI提高了查询的速度,减少了索引空间开销。但是LI的拟合误差较大,不支持插入等修改性操作。文中提出了一种利用梯度下降算法拟合数据的学习索引模型GDLIN(A Learned Index By Gradient Descent)。GDLIN利用梯度下降算法更好地拟合数据,减少拟合误差,缩短本地查找的时间;同时递归调用数据拟合算法,充分利用键的分布规律,构建上层结构,避免索引结构随着数据量而增大。另外,GDLIN利用链表解决LI不支持数据插入的问题。实验结果表明,GDLIN在无新数据插入的情况下,吞吐量是B+树的2.1倍;在插入操作占比为50%的情况下,是LI的1.08倍。

关键词:学习索引;梯度下降;拟合数据模型;链表
中图法分类号　TP391

GDLIN:A Learned Index By Gradient Descent

CHEN Shanshan[1],GAO Jun[2] and MA Zhenyu[3]
School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210003,China

Abstract　In the era of big data,data access speed is an important indicator to measure the performance of large-scale storage systems. Index is one of the main technologies to improve data access performance in database system. In recent years,learned index (LI) is proposed,which uses machine learning models instead of traditional B+-tree indexes,leverages pattern about the underlying data distribution to train the models and optimize the indirect search of data query into the direct search of function calculation,learned index can speed up queries and reduce the size of an index. However,the fitting effect of LI is general,and it assumes that the data is static and read-only,it does not support modification operations such as insertion. This paper presents GDLIN,a novel form of a learned index,which uses gradient descent algorithm to fit the data. Gradient descent algorithm can reduce the error between the predict position and the actual position,which can reduce the cost of local research. Besides,GDLIN recursive calls the construction algorithm until only one model is created,which makes full use of keys' distribution,and avoids the increase of the size of index with the data volume. In addition,GDLIN uses the sorted linked list to address the problem of data insertion. Experiment results demonstrate GDLIN improves the lookup throughput by 2.1× compared with the traditional B+-trees without insertion. Besides,GDLIN improves the lookup performance by 1.08× compared with the LI when the factor of insertion is 0.5.

Keywords　Learned index,Gradient descent,Fitting data model,Linked list

1　引言

索引是数据库中提升数据读取性能的技术之一。在传统索引技术中基于树结构的索引运用最为广泛,它们通过将数据的键组织成一棵树而减少查询的次数,从而提高查询的速度,但随着数据量的不断增加,树的体积会越来越大。所以近年来,研究人员也通过各种不同的方式减少索引的结构大小,例如前/后缀截断、哈夫曼编码等技术[1-2]。但是随着数据量的爆发式增加,这些方式也失去了效果,因为随着数据量的暴增,不仅键的数量会增加,同时键的长度也会增加,从而导致传统索引的开销也随之线性增加[3]。最近几年,人工智能和数据库得到了广泛的研究,进而有研究员将人工智能和数据库结合起来,利用人工智能技术解决传统数据库无法满足大规模数据库实例等问题[4]。而索引就是其中一个方面,在2018年,Kraska等人提出了"学习索引"的概念,将机器学习引入索引技术,有效地解决了索引空间开销问题,同时也提高了查询性能。

学习索引就是通过机器学习技术学习数据分布规律,通过模型拟合数据,将传统索引的间接查询优化为函数计算的直接查询,但是由于现实世界的数据不可能通过任何一个函数精准拟合,因此学习索引的数据拟合只是近似拟合,函数的计算预测是存在误差的,但即便如此,在可接受的误差范围内通过二分查找得到结果的效率还是要优于B+树等传统索引的性能。而且,学习索引只需要存储模型的参数即可,所以在

基金项目:国家自然科学基金面上项目(61972209)
This work was supported by the National Natural Science Foundation of China(61972209).
通信作者:陈珊珊(chenss@njupt.edu.cn)

空间代价上，学习索引也要优于传统索引。另外，因为之前传统索引的结构太大，所以大多都是基于磁盘而设计的，因而IO次数影响了数据查询性能，但是学习索引大幅度地减少了空间开销，在未来的研究中，可以考虑将整个索引缓存在内存中，提高数据的查询性能。

学习索引的具体实现涉及到多方面，首先要解决的就是数据拟合的问题。在数据拟合方面，虽然使用函数拟合数据分布的技术并不是一个新的研究方向，在之前就已经有了许多研究[5-14]。但这些技术并没有应用到索引中，所以对于学习索引来说，要运用这些技术，还要考虑一些额外问题。首先是模型的选择，高阶模型虽然可以更精准地预测数据，但是高阶模型的训练代价相当高。其次，因为学习索引的底层模型覆盖的数据集空间比B+树的叶节点覆盖的更大，所以如果模型的误差阈值较大，就会导致数据集上执行二分搜索的代价变高。因此，本文提出了利用梯度下降算法拟合数据分布的GDLIN，拟合数据的模型选择低阶函数，降低训练成本。该模型首先通过余弦相似度进行数据划分，将数据分布比较一致的数据划分到一起，之后利用梯度下降算法使拟合线段的误差进一步缩小，减少本地搜索的查询时间，提高查询的性能。

另外，大多数现有学习索引模型的上层结构还是使用常见的数据结构组织，如树、哈希表等。这些数据结构的最大缺点就是空间开销会随着数据量的增加而增大，它们没有充分利用学习索引的优势。GDLIN则是提取底层每个模型覆盖的第一个数据形成一个新的数据集，然后在该数据集上调用数据划分和数据拟合的算法产生上一层的模型，以这种递归的方式继续，直至最后一层只有一个模型产生。这样可以减少数据量对索引的影响，降低索引的空间开销。

因为插入新数据会违反模型预先保存的误差阈值，所以当有大量新数据插入时，会改变键的分布规律，导致模型失效，最终降低模型的查找性能，因此Kraska等[15]提出的学习索引模型(LI)并不支持数据插入。针对这个问题，GDLIN利用链表存放插入的新数据，在插入过程中保持链表中数据有序，在模型重训练时，可以利用快速排序的方法合并数据集，提高性能。

总结来说，本文提出的GDLIN主要有3方面的贡献，具体如下：

(1)利用余弦相似度对数据集进行划分，使每个子集中的数据分布尽可能一致，再利用梯度下降算法拟合数据分布减少误差，缩短本地查询的速度。

(2)递归调用数据划分和模型训练的算法，充分利用键的分布规律，构建上层结构，避免索引的空间开销随着数据量的增加而增大。

(3)利用链表存储插入的新数据，链表中数据保持有序，解决LI不支持插入的问题。

2 相关理论

2.1 B+树及其变种

传统索引中，B+树结构因其自身的优点而得到了最为广泛的应用。但随着数据量的暴增，B+树结构所占用的空间开销越来越大，影响了查询性能。事实上，近几年有研究[16]表明，在先进的内存DBMS中，为典型OLTP工作负载创建的索引最多可以消耗55%的可用内存。这种空间开销不仅影响了存储新数据的空间，还影响了处理中间数据的可用空间。为了解决这一问题，也有许多学者针对B+树结构进行研究。目前主要有两类优化方案：一类是各种不同的压缩方案，如前/后缀截断、字典压缩、哈夫曼编码[1-2,17-19]等，这些压缩方案的主要思想就是减少数据冗余和键的大小，但是在如今数据量快速增长的情况下，不仅键的数量在直线增长，键本身的长度也在增长；另一类是针对B+树的叶节点结构进行了修改，提出了A-树、BF-树、CSS-树[20-25]等，这些变体主要通过优化索引结构来尽可能地发挥硬件优势。

2.2 学习索引

然而，以上方案都是针对通用数据结构设计的，主要关注索引结构本身，因此忽略了数据的分布规律。而拟合数据分布可以优化绝大多数的索引结构，例如，简单的线性回归函数足以让系统存储和访问一组值为连续整数的键(例如从1到100M的键)，这在查找性能和内存开销方面比传统的B+树具有显著优势。由此可见，数据分布对提高存储系统的性能有显著作用。但是在现实生活中，提前精准获取数据的分布规律几乎是不可能的，甚至有的数据分布模式十分复杂，从未见过。所以Kraska等提出了利用机器学习训练模型来拟合数据分布，从而产生了“学习索引”这一概念。

之后针对学习索引的精度问题，Kraska等提出了递归模型索引(Recursive Model Index，RMI)。RMI结构受到了混合专家模型[26]的启发，它的主要思想是建立一个模型层次结构，在每个阶段，上层模型根据中间预测结果选择下层的一个模型，直到最后阶段预测位置[5]，图1是LI的RMI结构图。RMI可以在构建时混合使用多种模型，因此可以充分利用不同模型的优势。但是LI的RMI结构需要提前设置模型数量、误差阈值等，这些数值的大小会影响整个LI结构，模型数量设置过大会影响数据查询性能，模型数量设置过小会影响数据的拟合效果。误差阈值的设置也有类似问题。此外，LI并不支持数据插入，因为新数据的插入会影响数据分布规律，进而引起模型重训练，从而产生额外的成本开销，影响整个索引结构的性能。

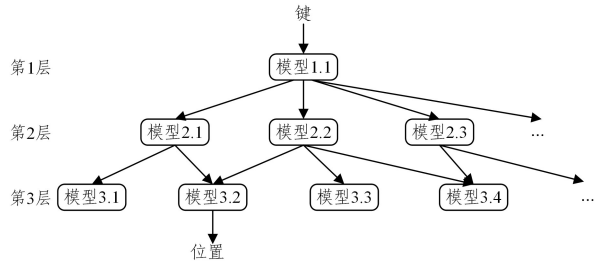


图1 RMI结构

Fig. 1 Structure of RMI

针对上述有关LI的问题，Kraska等也研究了新的索引结构——FITing-Tree[3]。FITing-Tree使用分段线性函数进行数据拟合，通过锥算法，根据预先设定的误差阈值判断数据点是否在同一个锥内，若添加的新数据点在锥外即作为一个新线段的起点，最后记录每条段的起点和斜率。这种方式限制了搜索的最差性能，其结构图如图2所示。FITing-Tree的

上层结构使用的是 B+树,避免了模型数量的设置,但是这并没有充分利用学习索引的优势,随着数据量的增大,上层结构的空间开销依然会越来越大。

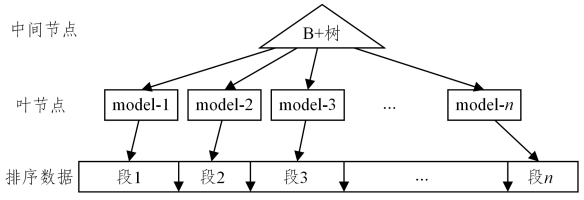


图 2 FITing-Tree 结构

Fig. 2 Structure of FITing-Tree

3 设计

GDLIN 主要改进了 LI 中的数据拟合部分,其使用余弦相似度进行数据划分,再通过梯度下降算法进行数据拟合,减少拟合误差,提升数据查询性能。其次,GDLIN 的上层结构是递归调用底层拟合数据的算法构建,充分利用了学习索引的优势,在进一步提升数据查询性能的同时也减少了索引的空间开销。针对数据插入问题,GDLIN 利用链表结构,通过模型预测键的插入位置,然后根据位置的状态选择不同的处理方式将数据插入。

3.1 数据划分

目前大多数学习索引模型都是通过欧几里得距离作为相似度进行数据划分,但是欧几里得距离体现的是个体数值的绝对差异,多用于需要从维度的数值大小中体现差异的分析中;而对于学习索引中的数据拟合,我们更多的是需要将在同一条线段附近的数据划分在一起,其中更重要的是方向上的差异,而对绝对数值的差异并不敏感。所以 GDLIN 采用余弦相似度进行数据划分,当余弦值越接近 1 时,表示 3 个点越接近于同一条直线。每添加一个新的点时,该点与前面两个点形成两条向量,计算这两条向量夹角的余弦值,当余弦值小于预先设定的阈值时,说明这两条向量接近正交,该点就作为分段点。由此可以产生多个分段点,进而将数据集分成多个子数据集,之后再利用梯度下降算法在这些数据集上拟合数据。如图 3 所示,由于点 p_5 和点 p_{11} 与其前两个点形成的向量夹角余弦值小于预先设定的阈值,因此 p_5 和 p_{11} 作为两个分段点,将图 3 中的 13 个点分为了 3 个子区域。

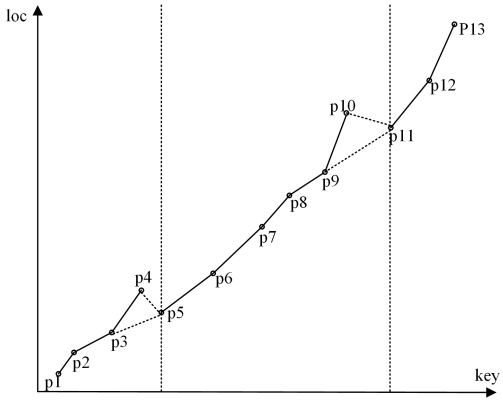


图 3 数据的余弦相似度

Fig. 3 Cosine similarity of data

算法 1 给出了根据余弦相似度进行划分的过程。

算法 1 余弦值划分数据

输入:数据集 dataset[N],阈值 threshold

输出:分段点数组 split_index[ ]

```
1. DATA_PARTITION(dataset)
2.   if dataset < 3 then
3.   /* 当数据集的数量小于 3 时直接返回 */
4.     return
5.   else
6.     for i ← 2 to N-1
7.     do v1 ← dataset[i-1]-dataset[i]
8.       v2 ← dataset[i-2]-dataset[i]
9.       cos ← cosine(v1,v2)
10.    if cos < threshold then
11.      split_index[ ] ← cos
12.  return split_index
```

3.2 数据拟合

FITing-Tree 中数据拟合通过堆算法判断数据点的添加是否会破坏之前点的误差约束,若破坏了,则该点作为新段的起点,重新调用堆算法。每段的斜率取最大斜率和最小斜率之和的二分之一。这种算法计算简单,但是拟合的效果一般,每条段内的误差较大。段的误差即段内的每个点距离其实际位置的距离的最大值,当误差越大时,在进行本地搜索时消耗的时间就会越长,进而影响查询效率。GDLIN 为了减少误差值,选择使用梯度下降拟合数据。因为 GDLIN 中依然选择使用线段拟合,所以其计算误差最小值的过程如下:

假设线段的公式为:

$$h(x)=s*x+i \tag{1}$$

式(1)的损失函数为:

$$loss=\sum(h(x)-y) \tag{2}$$

式(2)中 y 为数据的实际位置,$h(x)$为函数预测的位置。

梯度下降算法的具体过程:

(1)求损失函数的偏导数

$$\frac{\partial loss}{\partial s}=2*x*(s*x+i-y) \tag{3}$$

$$\frac{\partial loss}{\partial i}=2*(s*x+i-y) \tag{4}$$

(2)将样本点带入式(1)和式(2)中更新参数

$$s=s-\frac{\partial loss}{\partial s}*learning\_rate \tag{5}$$

$$i=i-\frac{\partial loss}{\partial i}*learning\_rate \tag{6}$$

(3)根据式(5)和式(6)计算出的新的 s 和 i 值计算新的 $loss$ 值

通过式(5)和式(6)可以看出,求解的一个关键点就是 $learning\_rate$ 的初始值。$Learning\_rate$ 是梯度下降算法中的步长,每当计算出下降方向时,需决定步长大小,而步长的大小也会影响数据拟合的性能。步长过大时,无法拟合到最小值;步长过小时,拟合速度太慢,影响性能。

梯度下降算法中除了 $learning\_rate$ 是一个重要的影响因素外,参数迭代次数 $iterator\_num$ 也是重要的影响因素。迭代次数在确定步长的情况下,会影响最终能取得的最小误差,当迭代次数达到一定值之后,拟合误差就会接近最小值。

算法 2 给出了利用梯度下降算法拟合线段的过程。

算法 2 下降梯度拟合数据

输入：子数据集 sub_data[N]，起始斜率 slope，起始截距 intercept，步长 learning_rate，迭代次数 iterator_num

输出：拟合线段的斜率 s 和截距 i

```
1. /* 梯度下降拟合数据 */
2. GRADIENT_DESCENT( sub_data,slope,intercept,learning_rate,
   iterator_num )
3.     s ← slope
4.     i ← intercept
5.     for j ← 0 to iterator_num−1:
6.         do s,i ← GRADIENT_STEP(s,i,sub_data,learning_rate)
7.     return [s,i]
8. /* 每次迭代更新参数 */
9. GRADIENT_STEP(now_s,now_i,data,learning_rate)
10.    s ← 0
11.    i ← 0
12.    for j ← 0 to N−1
13.    do x ← data[i][0]
14.       y ← data[i][1]
15.       /* 求偏导数 */
16.       s ← 2 * x *(now_s * x+now_i-y)
17.       i ← 2 *(now_s * x+now_i-y)
18.    /* 更新参数 s,i */
19.    s ← s / N
20.    i ← i / N
21.    new_s ← now_s-learning_rate * s
22.    new_i ← now_i-learning_rate * i
23.    return [new_s,new_i]
```

最后提取底层每个模型所覆盖的第一个数据，产生一个新的数据集，然后在该数据集上递归调用数据划分算法和数据拟合算法，形成上层模型，递归此操作直至最后只产生一个模型作为最上层。

3.3 数据插入

GDLIN 通过一组链表支持数据插入，每当有新数据插入时，先通过模型计算出插入的位置，若位置为空则直接插入，若位置不为空，则 GDLIN 会为小于它的最大键分配一个对应的链表，并将数据添加到该链表中，并且链表中的数据也要保持有序。这样在之后模型重训练时，可以通过排序算法快速进行数据合并。如图 4 所示，当我们插入 24 时，因为根据模型计算出的位置不为空，所以找到小于它的最大键 19 所对应的链表，将 24 插入到链表中，并且链表中依旧保持有序。另外，每个链表大小与一个缓存条的大小一样，这样可以减少 IO 次数，提高之后的查找效率。

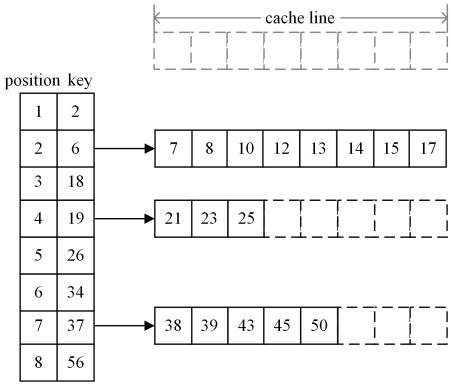


图 4 插入数据过程

Fig. 4 Insert data process

但如果不断地分配链表，GDLIN 的查询性能会下降，因为 GDLIN 必须花费大量时间在链表上搜索。因此，当链表达到一定长度时，再插入新数据，就要进行重训练。首先我们要将链表中的数据与之前的数据集进行合并，因为链表中的数据也是保持有序的，所以可以很快地获取合并后的数据集；然后在该数据集上调用数据划分算法，判断该数据集是否因为插入的新数据改变了数据分布规律，若改变了则会被划分为多个数据子集，在这些数据子集上调用数据拟合算法，产生多个段，将新产生的段所覆盖的第一个数据插入到上一层对应的模型中。若是上一层对应的模型覆盖的数据集也达到了阈值，则调用前面的操作，先进行数据划分，再执行数据拟合过程。

4 实验结果与分析

为了测试 GDLIN 的性能，本文将其与 B+树、LI、FITing-Tree 进行对比。其中 B+树选择扇出为 128 的结构，因为在文献[15]中描述了 B+树在这种配置下取得了最好的查找性能。另外，因为 LI 的模型数量需要提前设置，所以选择底层模型数为 10K 和 50K 的 LI 进行比较。我们使用不同的数据集进行测试，分别为 Lognormal 人工数据集和 Weblogs 真实数据集。Lognormal 数据集包含 1.9 亿个遵循对数正态分布的唯一值；Weblogs 数据集包含 2 亿条日志条目，使用时间戳作为索引。然后我们通过 YCSB[27] 生成两种工作负载，一种是只读的，一种是插入因子为 0.5 的。

图 5 给出了在无插入的情况下，GDLIN 和 B+树、LI-10K、LI-50K、FITing-Tree 在两个数据集上的吞吐量。横轴表示测试的数据集，纵轴表示每秒的吞吐量。这里我们还引入了二分查找算法，因为这种方式可以预测粒度等于整个数据集大小的最坏情况。从图 5 中可以看到，GDLIN，FITing-Tree，LI 在查询性能上均优于 B+树。这是因为 GDLIN，FITing-Tree 和 LI 都通过学习数据中的分布规律，训练出适用于当前数据集的模型，从而在查询数据时，能以更快的时间查询到记录的位置，而 B+树的查询性能固定为 log(N)。在 Lognormal 数据集上，GDLIN 的查询吞吐量是 B+树的 2.1 倍；在 Weblogs 数据集上，GDLIN 的查询吞吐量也是 B+树的 2.1 倍。另外可以看出，在两个数据集上，GDLIN 性能也都优于 FITing-Tree，这是因为 GDLIN 的数据划分更好，数据拟合误差更小，因此本地搜索的时间更少。而 GDLIN 与 LI-10K 的查询性能相近，但比 LI-50K 略差，这是因为 LI-50K 的底层模型数量设置地更多，数据进行拟合的效果更好，但是模型量并不是越多越好。一方面，模型数量过多时，需要维护的成本就越高；另一方面，模型数量设置太大时，会存在很多冗余的模型，造成资源浪费。

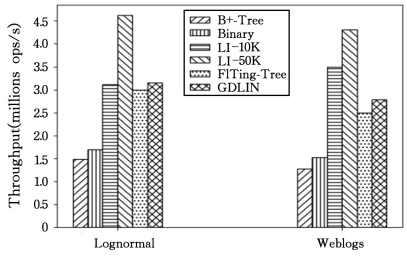


图 5 无插入查询吞吐量

Fig. 5 Lookup throughput without inserts

图 6 给出了在插入操作占比为 50%的情况下,GDLIN 和 B+树、LI、FITing-Tree 在两个数据集上的吞吐量。明显可以看到 LI 的性能不如无插入时的吞吐量,这是因为 LI 通过增量索引解决插入问题,必须在每次查询中查找两种结构,影响了整个查询性能。另外,因为插入新数据会改变数据的分布规律,引起模型的重训练,所以 GDLIN 的查询效果并没有优于 B+树,但是 GDLIN 通过链表实现了数据可扩展性,优化了 LI 的插入查询的问题,并在两个数据集上均略优于 FITing-Tree。在有新数据插入的情况下,GDLIN 的查询性能是 LI 的 1.08 倍,是 FITing-Tree 的 1.03 倍。

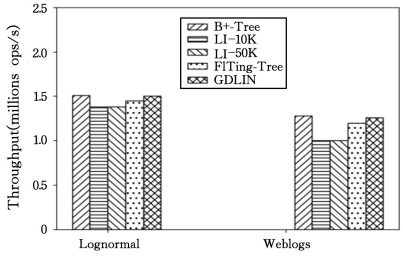


图 6 插入因子为 0.5 的查询吞吐量

Fig. 6 Lookup throughput with an insert factor of 0.5

图 7 给出了 GDLIN,LI,FITing-Tree 和 B+树的空间开销。与传统索引相比,学习类型的索引空间开销更小,这是因为 B+树结构需要存储节点中的所有数据,而学习索引可以通过模型管理数据,只需要存储模型的参数即可,大大减少了空间开销。由图 7 可看出,学习索引的空间开销要比 B+树少了 15～35 倍。

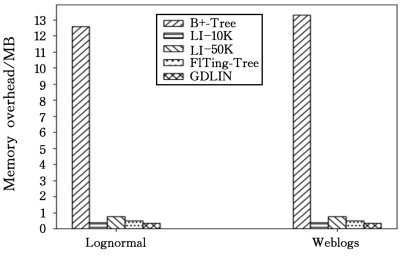


图 7 索引空间开销

Fig. 7 Index space overheads

图 8 通过平均误差展示 GDLIN 和 FITing-Tree 中的线段拟合算法的效果差异。FITing-Tree 在拟合数据时,只是通过最大斜率和最小斜率求平均值作为线段的斜率,所以拟合效果一般。GDLIN 通过实验发现在 *learning_rate* 为 0.0001,*iterator_num* 为 100000 时,下降梯度算法的拟合效果接近最佳。通过图 8 可以发现,在多个数据集中梯度下降算法的拟合效果均优于 FITing-Tree 的拟合效果。

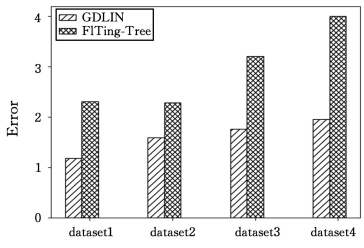


图 8 平均误差

Fig. 8 Average error

结束语 设计一种高效的索引结构一直是数据库领域的研究热点,Kraska 等提出的结合机器学习的学习索引,为索引研究打开了新方向,对比 B+树等传统索引,取得了显著的效果。但是其本身还存在许多不足,所以本文提出了基于梯度下降算法的 GDLIN,它可以在支持数据插入的情况下提高数据查询性能。其创新点之一在于利用余弦相似度进行数据划分,划分效果更好;另一个创新点在于其利用梯度下降算法进行数据拟合的方式,与简单的平均值计算一线段相比性能更高,因为它尽可能地减少了拟合误差,进而减少了本地查询的时间开销。在后续研究中,可以尝试选择不同的模型进行训练,进一步提高查询的性能,并且在模型解耦方面进行研究,减少模型重训练的时间成本。

参 考 文 献

[1] BAYER R,UNTERAUER K. Prefix B-trees[J]. ACM Transactions on Database Systems,1977,2(1):11-26.

[2] STONEBRAKER M. The Case for Partial Indexes[J]. SIGMOD Record,1989,18(4):4-11.

[3] GALAKATOS A,MARKOVITCH M,BINNIG C,et al. FITing-Tree:A Data-aware Index Structure[C]// The 2019 International Conference. 2019:1189-1206.

[4] ZHOU X,CHAI C,LI G,et al. Database meets artificial intelligence:a survey[J]. IEEE Transactions on Knowledge and Data Engineering,2020,34(3):1096-1116.

[5] CHEN L,GAO Y,LI X,et al. Efficient metric indexing for similarity search and similarity joins[J]. IEEE Transactions on Knowledge and Data Engineering,2017,29(3):556-571.

[6] FU T C. A review on time series data mining[J]. Engineering Applications of Artificial Intelligence,2011,24(1):164-181.

[7] EICHINGER F,EFROS P,KARNOUSKOS S,et al. A time-series compression technique and its application to the smart grid [J]. VLDB Journal,2015,24(2):193-218.

[8] ESCH R E,EASTMAN W L. Computational methods for best spline function approximation[J]. Journal of Approximation Theory,1969,2(1):85-96.

[9] KEOGH E,CHU S,HART D,et al. An online algorithm for segmenting time series[C]// Proceedings 2001 IEEE International Conference on Data Mining. IEEE,2002.

[10] LIU XY,LIN Z,WANG H. Novel online methods for time series segmentation[J]. IEEE Transactions on Knowledge & Data Engineering,2008,20(12):1616-1626.

[11] NEUMANN T,MICHEL S. Smooth interpolating histograms with error guarantees[C]// British National Conference on Databases. 2008.

[12] SHATKAY H,ZDONIK S B. Approximate queries and representations for large data sequences[C]// ICDE. 2019:536-545.

[13] XU Z,ZHANG R,RAMAMOHANARAO K,et al. An adaptive algorithm for online time series segmentation with error bound guarantee[C]// EDBT. 2012:192-203.

[14] WANG L,WANG W J,JIANG G X. Optimization for Smoothing Parameter in Process of Data Fitting[J]. Computer Science,2015,42(9):226-229.

[15] KRASKA T,BEUTEL A,CHI E H,et al. The case for learned index structures[C]// Proceedings of the International Conference on Management of Data. 2018:489-504.

[16] ZHANG H,ANDERSEN D G,PAVLO A,et al. Reducing the

storage overhead of main-memory OLTP databases with hybrid indexes[C]// International Conference on Management of Data. ACM, 2016.

[17] ZUKOWSKI M, HEMAN S, NES N, et al. Super-scalar RAM-CPU cache compression[C]// International Conference on Data Engineering. IEEE, 2006.

[18] BÖHM M, SCHLEGEL B, VOLK P B, et al. Efficient In-Memory indexing with generalized prefix tees[C]// DBLP. 2011.

[19] RAO J, ROSS K A. Cache conscious indexing for Decision-Support in main memory[C]// Proceedings of the International Conference on Very Large Data(VLDB). Morgan Kaufmann/ACM, 1999: 78-89.

[20] GALAKATOS A, MARKOVITCH M, BINNIG C, et al. A-Tree: a bounded approximate index structure[J]. arXiv: 1801.10207, 2018.

[21] ATHANASSOULIS M, AILAMAKI A. BF-Tree: approximate tree indexing[C]// International Conference on Very Large Databases. 2014.

[22] HWANG D, KIM W H, WON Y, et al. Endurable transient inconsistency in byte-addressable persistent B+-tree[C]// 16th USENIX Conference on File and Storage Technologies. 2018: 187.

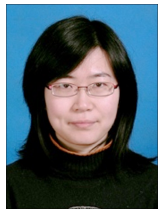
[23] RAO J, ROSS K A. Making B+-trees cache conscious in main memory[C]// Proceedings of the Conference on Management of Data(SIGMOD). ACM, 2000: 475-486.

[24] CHEN S, GIBBONS P B, MOWRY T C. Improving index performance through prefetching[J]. ACM SIGMOD Record, 2002, 30(2): 235-246.

[25] SHAZEER N, MIRHOSEINI A, MAZIARZ K, et al. Outrageously large neural networks: the sparsely-gated mixture-of-experts layer[C]// Proceedings of the International Conference. on Learning Representations. 2017.

[26] BRAESS D. Chebyshev approximation by spline functions with free knots[J]. IMA J NUMER ANAL, 1992, 12(5): 357-366.

[27] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems withYCSB[C]// Proceedings of the 1st ACM Symposium on Cloud Computing. 2010: 143-154.



CHEN Shanshan, born in 1980, Ph. D, associate professor. Her main research interest is large-scale distributed storage systems and architectures.