Научная статья

УДК 004.651
DOI 10.25205/1818-7900-2025-23-3-67-78

Анализ и тестирование индекса RadixSpline

Илья Максимович Стецкий
Борис Николаевич Пищик

Новосибирский государственный университет
Новосибирск, Россия

i.stetskii@g.nsu.ru
b.pishchik@g.nsu.ru

Аннотация

Статья посвящена тестовой реализации индекса RadixSpline и исследованию его производительности при различных параметрах настойки.

Индекс RadixSpline относится к категории обученного индекса и предназначен для эффективного поиска в отсортированных данных в структуре ключ – значение. Индекс строится за один проход по данным и состоит из сплайн-аппроксимации с ограниченной погрешностью (GreedySpline) и разреженного индекса (RadixTable).

В работе перечислены известные обученные индексы и сравнение их с исследуемым RadixSpline, который выгодно отличается от других тем, что имеет простую структуру и малое количество параметров настройки. Проведен анализ работы RadixSpline и его частей, а также влияние параметров на производительность и объем затрачиваемой памяти. Тестирование реализации показало, что RadixSpline способен значительно сократить время поиска по сравнению с бинарным поиском. Кроме того, при других настройках можно сократить используемую память. В работе предлагаются уточнения в алгоритме RadixSpline и исследование возможности его интеграции в промышленные продукты для работы с большими данными.

Ключевые слова

RadixSpline, Learned Index, сплайн-аппроксимация, разреженный индекс, оптимизация поиска, производительность СУБД, машинное обучение в индексации

Финансирование

Исследование выполнено при поддержке благотворительного фонда содействия образованию и становления молодых специалистов «Высшая лига».

Для цитирования

Стецкий И. М., Пищик Б. Н. Анализ и тестирование индекса RadixSpline // Вестник НГУ. Серия: Информационные технологии. 2025. Т. 23, № 3. С. 67–78. DOI 10.25205/1818-7900-2025-23-3-66-78

Analysis and testing of the RadixSpline Learned index

Ilya M. Stetskii, Boris N. Pischik

Novosibirsk State University,
Novosibirsk, Russian Federation

i.stetskii@g.nsu.ru
b.pishchik@g.nsu.ru

Abstract

Recent advances in machine learning have introduced a novel approach to database indexing known as learned indexes. Among these, the RadixSpline index has emerged as a particularly promising solution for efficient search in sorted key-value data. This paper presents a comprehensive analysis and performance evaluation of the RadixSpline index, which combines spline approximation with bounded error (GreedySpline) and a sparse radix table structure.

We investigate the index's behavior under different parameter configurations, focusing on two key parameters: the spline approximation error (err) and the radix prefix length (r). Our experimental study, conducted on a 25 MB subset of the MovieLens32M dataset, demonstrates that RadixSpline can achieve up to 30% faster search times compared to traditional binary search while reducing memory usage by over 80 % compared to RocksDB's sparse index implementation.

The study makes several important contributions:

(1) we provide a detailed analysis of parameter sensitivity, identifying optimal configurations for various use cases;

(2) we address implementation challenges not covered in the original work, including handling of non-contiguous prefixes and missing keys;

(3) we demonstrate the index's suitability for real-time applications through its single-pass construction property.

Our results confirm that RadixSpline offers significant advantages over conventional indexing methods, particularly in scenarios requiring fast searches with limited memory resources. The findings suggest strong potential for integrating RadixSpline into production database systems, with RocksDB being a particularly promising candidate for such integration.

Keywords

RadixSpline, Learned Index, spline approximation, sparse index, search optimization, DBMS performance, machine learning in indexing

Funding

The work was supported by the charitable foundation for the promotion of education and formation of young specialists 'Высшая лига'.

For citation

Stetskii I. M., Pischik B. N. Analysis and testing of the RadixSpline Learned index. *Vestnik NSU. Series: Information Technologies*, 2025, vol. 23, no. 2, pp. 67–78 (in Russ.) DOI 10.25205/1818-7900-2025-23-3-67-78

Введение

В 2017 г. был предложен новый подход к индексации, использующий идеи машинного обучения [1]. Одним из перспективных индексов, созданных в рамках этого подхода, является индекс RadixSpline [2]. Это обученный индекс, разработанный специально для поиска в отсортированных данных.

В индексе используются две структуры: сплайн-аппроксимация с ограниченной погрешностью (ошибкой), представляющая исходные данные в виде набора прямых (GreedySpline [3]), и разреженный индекс, который содержит индексы на часть набора прямых. RadixSpline достаточно прост в реализации и имеет минимальное количество параметров настройки: погрешность аппроксимации и размер разреженного индекса. RadixSpline строится за один проход по данным, что крайне важно для потоковой обработки данных в реальном времени, и работы с большим объемами информации.

В работе исследуется производительность тестовой реализации индекса RadixSpline в зависимости от параметров настройки. Тестирование реализации проводилось на датасете

MovieLens32M [4]. Показатели производительности сравнивались с разреженным индексом [5] RocksDB [6], на файлах данных размера 25 Мб.

СУБД RocksDB использует LSM-дерево [7], хранит данные в файлах Sorted String Table (SST-файлах) и использует *бинарный поиск*, который на больших данных может потребовать значительного времени. Целью работы является исследование эффективности структуры, предоставленной авторами статьи [2], для использования ее в RocksDB вместо LSM-дерева.

На момент выполнения исследования не найдено публикаций об использовании RadixSpline в промышленных продуктах и выполненная работа – шаг в этом направлении.

Обзор обученных индексов

Концепция обученных индексов (Learned Index), рассматривающая индексы как модель и предлагающая использовать методы машинного обучения, опубликована в [1].

Концепция базируется на представлении индекса как модели поиска данных, организованных определенным образом [8]. Например, B-дерево можно представить как модель, которая отображает ключ на позицию записи в отсортированном массиве, а хеш-индекс – как модель, которая отображает ключ на позицию в неотсортированном массиве. Основная концепция обученных индексов заключается в том, что модель может «научиться» сортировке или структуре ключей и использовать это для эффективного предсказания позиции искомых данных.

В статье [1] авторы показали, что по сравнению с оптимизированными B-деревьями использование нейронных сетей позволяет улучшить производительность поиска на 70 % и сэкономить память почти в 10 раз. Что дало начало исследованию данной тематики.

В статье [9] делается обзор последних исследований в области Learned Index. Рассматриваются и описываются различные структуры. Самой знаковой из них считается Recursive Model Index (RMI), рекурсивно использующая модели на нескольких уровнях. На каждом уровне модель выбирает дочернюю модель на основе входного ключа, пока не достигнет листовой модели, которая предсказывает фактическую позицию ключа в отсортированном массиве. RMI был подробно проанализирован и рассмотрен в статье [10]. Авторы исследуют влияние каждого гиперпараметра на производительность, показывают, что в дополнение к типам моделей и размеру слоя для достижения наилучшей возможной производительности необходимо учитывать границы ошибок и алгоритмы поиска.

В процессе исследований была рассмотрена группа гибридных Learned Index, оптимизирующих традиционные индексы вспомогательными моделями. Такие индексы используют модели машинного обучения для улучшения производительности поиска, в то время как традиционные структуры данных обеспечивают надежность поиска.

Ниже приведена краткая характеристика рассмотренных гибридных Learned Index:

1. Hybrid_RMI [8]. Объединяет традиционное B-дерево с RMI [1] структурой для создания гибридной индексной структуры.
2. PLEX [11]. Использует традиционный Hist-tree [12] для построения обученного индекса с одним настраиваемым параметром.
3. LSI [13]. Использует также компактный Hist-tree для создания обученного индекса для несортированных данных.
4. ShiftTable [14]. Вспомогательный алгоритмический слой, который фиксирует распределение данных на микроуровне и устраняет локальные погрешности выученной модели, затрачивая не более одного обращения к памяти.
5. LBF [15]. Представляет структуру обученного фильтра Блума, объединяя ML-модели с традиционными фильтрами Блума.

6. RadixSpline (RS) [2]. Использует разреженный индекс и линейную сплайн-аппроксимацию для построения однопроходного обученного индекса с низкими вычислительными затратами.
7. RSS [16]. Расширяет методы RS для индексации строк.

Для достижения поставленной цели выбран индекс RadixSpline, который позволяет строить индекс за один проход по данным. Hybrid_RMI строится рекурсивно, для построения Histtree, используемых в PLEX и LSI, также необходимы дополнительные проходы. Кроме того, RadixSpline проще в настройке из-за меньшего числа параметров, требует меньше вычислительных ресурсов, довольно компактный и поддерживает диапазонные запросы, в отличие от LBF и RSS.

Обзор RadixSpline

Обученный индекс RadixSpline опубликован в статье [2] в 2020 г. Он предназначен для сопоставления ключа поиска и его индекса (положения ключа в базе данных) в отсортированных по возрастанию ключа исходных данных.

Структура RadixSpline состоит из двух компонентов: набора сплайнов (результат сплайн-аппроксимации, в статье используется GreedySplineCorridor) и разреженного индекса, называемого RadixTable.

Метод аппроксимации GreedySplineCorridor

Методы сплайн-аппроксимации GreedySpline опубликован в 2004 г. [3]. В статье рассматривается несколько вариантов сплайн-аппроксимации. Для RadixSpline будет использоваться метод GreedySplineCoridor.

Для построения GreedySplineCorridor необходима заданная погрешность аппроксимации (ошибка) err, которую необходимо подбирать для различных задач. Данный параметр является одним из двух параметров настройки RadixSpline

Алгоритм построения сплайна (GreedySplineCorridor)

Входные данные алгоритма – массив отсортированных данных **S**.

В алгоритме входные данные представляются как точки S_i на плоскости с координатами *(key_i, i)*, где *key_i* – значение элемента массива (ключ), *i* – порядковый номер элемента данных (индекс ключа), $i = 1, ..., n$.

Обозначим заданную погрешность аппроксимации **err**. Первую точку данных называем точкой начала сплайна **base_point (key_1, 1)**.

По первым двум точкам данных S_1, S_2 строим прямую. Находим два критических угла, которые задают границы допустимого диапазона. Пусть **p –** функция нахождения тангенса угла между прямой, заданной двумя точками, и горизонтом:

$$P(i, j) = \frac{y_j - y_i}{x_j - x_i}. \quad (1)$$

В расчетах используется тангенс, так как он позволяет удобно сравнивать углы наклона. Если тангенс одного угла больше второго, значит прямая проходит выше. Тогда

p_max = p(base_point, (key_2,2+err)), **p_min=p(base_point, (key_2,2-err)).**

Далее точки добавляются по одной в цикле.

ЦИКЛ

При добавлении новой точки **(key_i, i)** строится прямая из начала сплайна **base_point** до новой точки, и проверяется ***условие вхождения прямой*** в допустимый диапазон:

(p_max ≥ p(base_point, (key_i, i)) ≥ p_min).

ЕСЛИ ***условие вхождения прямой*** выполняется,

ТО

- переобозначаем критические углы **p_max=min(p_max, p(base_point, (key_i, i+err))), p_min=max(p_min, p(base_point, (key_i, i-err)))**
- **Переход в начало цикла**

ИНАЧЕ

- Прямая аппроксимации считается построенной до предыдущей точки (base_point, S_(i-1)).
- Создаем новую прямую. Предыдущую точку считаем новой базовой, из нее находим критические углы к S_i:

p_max = p(S_(i-1), (key_i, i+err)), p_min=p(S_(i-1), (key_i, i-err)).

- **Переход в начало цикла**

В результате работы создается массив прямых (сплайнов), описывающих исходные данные. Сложность построения *GreedySplineCorridor* – O(1). Проверка каждого элемента происходит за константное время, расходуемое на расчет тангенса угла по формуле (1) и на два сравнения рациональных чисел. В каждой ветви условия внутри цикла также идет расчет тангенса угла по формуле (1) и сравнение двух рациональных чисел.

Поиск по GreedySplineCorridor

Пусть нам дан ключ x, и нам необходимо найти такое $S(x)$, что реальный индекс ключа, $p(x)$ будет лежать в границах $S(x) - e <= p(x) <= S(X) + e$.

Для оценки, нам необходимо найти две идущие подряд точки сплайна (k, p) такие, что $k_{left} <= x <= k_{richt}$. В таком случае, $S(x)$ находится по следующей формуле:

$$S(x) = p_{left} + (x - k_{left}) * \frac{p_{right} - p_{left}}{k_{right} - k_{left}}. \quad (2)$$

Построение разреженного индекса RadixTable

Построение RadixTable, по описанию в [2], происходит за один проход по отсортированным по возрастанию ключа точкам сплайна. Каждый раз, когда встречается сплайн с отличным от предыдущего r-битным префиксом, в нужную ячейку вставляется указатель на данную точку сплайна. То есть в ячейку с номером N помещается указатель на первый сплайн (Spline point), значение r-бит префикса которого равно N. Длина префикса задается как параметр структуры при ее создании.

Для RadixTable выделяется массив размером 2^r. Больший размер префикса позволяет получить бóльшую точность, но и требует больше памяти под таблицу.

Заметим, что на рис. 1 приведен пример RadixTable, в которой ячейки заполнены непрерывной последовательностью целых чисел. Авторы не рассматривают случай, в котором значения префиксов не являются непрерывной последовательностью и не сообщают, как заполняется ячейка таблицы K, если нет точки сплайна со значением префикса, равным K. В тестовой реализации это учтено.

Поиск элементов внутри RadixSpline

Рассмотрим пример (рис. 1), представленный в статье, и алгоритм авторов, где введены следующие обозначения:

- Lookup key – искомый ключ,
- Radix table – массив элементов разреженного индекса,
- Pointer – указатели на определенные точки сплайна, хранящиеся в RadixTable,
- CDF – cumulative distribution function, в оригинальной статье рассматривалась кумулятивная функция распределения для построения GreedySplineCorridor. В контексте RadixSpline вместо CDF используется набор отсортированных по возрастанию исходных данных.

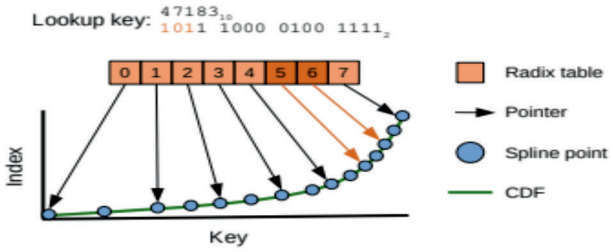

Рис. 1. Пример поиска ключа с использованием структуры RadixSpline, при $r = 3$
Fig. 1. Example of key search using the RadixSpline structure, with $r = 3$

Рассмотрим рисунок подробнее. Необходимо найти ключ 47183. Битовое представление данного ключа – 1011 1000 0100 1111. При создании структуры было решено использование префикса $r = 3$. Выбираем 3 первых значения битового представления (101), что в десятичном представлении – 5. По алгоритму из [2] искомый ключ должен находиться не ранее элемента, который в RadixTable лежит в ячейке с индексом 5. Кроме того, он должен находиться не правее элемента, лежащего в ячейке следующей ячейки, т. е. ячейки с индексом 6.

Однако, учитывая замечание в предыдущем пункте, префикс искомого ключа может указывать на ячейку таблицы, не содержащую точку сплайна. И необходимо определять точку сплайна, являющуюся началом диапазона поиска. В тестовой реализации это уточнено.

Итоговый алгоритм поиска индекса ключа с использованием RadixSpline:

1) в ключе, индекс которого нужно найти, выделяем битовый префикс длины r;
2) используя RadixTable, находится интервал поиска по Spline point;
3) бинарным поиском находится пара Spline point, между которыми лежит искомый ключ;
4) по формуле (2) находится индекс S(x), который отличается от искомого индекса не более чем на e;
5) бинарным поиском на интервале {S(x)-e, S(x)+e} находится искомый ключ, если ключа на этом интервале нет, значит его нет во всем наборе данных.

Зависимость времени работы и занимаемой памяти от заданных параметров структуры

Занимаемая структурой память зависит напрямую от количества используемых бит префикса (параметра r), и от количества точек в GreedySplineCorridor, определяемых погрешностью. Чем меньше погрешность, тем больше точек сплайна необходимо построить для аппроксимации.

Затраченное время для поиска в основном складывается из времени бинарного поиска по точкам GreedySplineCorridor, а также бинарного поиска внутри интервала погрешности. Очевидно, что с уменьшением погрешности интервал для второго поиска будет уменьшаться, но в то же время количество точек GreedySplineCorridor будет увеличиваться, как и время поиска в нем. При этом увеличение r, количества бит префикса, будет сокращать время поиска, уменьшая интервалы между указателями RadixTable.

Таким образом, можно утверждать, что с увеличением r экспоненциально растет затрачиваемая память (так как размер RadixTable равен 2^r) и при этом сокращается время поиска. Увеличение размера погрешности сокращает размер занимаемой памяти, уменьшает время финального бинарного поиска, но увеличивает время первого бинарного поиска.

Тестовая реализация индекса RadixSpline

Для проведения экспериментов и исследования описанных выше зависимостей написана программа на языке Python. В основном она использует алгоритмы, описанные в [2; 3]. Уточнения алгоритма, реализованные в программе, описаны в особенностях реализации. Главными компонентами ее являются два класса: GreedySpline и RadixSpline.

GreedySpline реализован потоковым методом: для его работы не требуется хранение в памяти всего набора данных, он обрабатывает каждое значение по очереди, сохраняя только итоговый результат. Такой подход позволяет работать с данными любого размера.

Для построения RadixSpline исходные данные не используются. RadixSpline строится только по аппроксимации GreedySpline. При построении разреженного индекса RadixSpline необходимо знать битовую длину последнего элемента в GreedySpline. Используя битовую длину максимального элемента, находят битовый префикс всех элементов GreedySpline. В том случае, когда ключи имеют разную длину, короткие ключи дополняются нулями до необходимой длины.

Особенности реализации

1. *Длина каждого сплайна > err*.

Ошибка аппроксимации **err** означает, на какое количество индексов мы можем ошибиться при поиске элемента на прямой сплайна. Так как индексы элементов хранятся без пропусков, можно утверждать, что для любого значения ключа с индексом k, ключи с индексами $k + 1$, …, $k +$ е будут лежать на прямой аппроксимации. Этим мы можем ограничить сверху количество точек сплайна для любого набора данных A как $\lceil |A|/e \rceil + 1$.

2. *Поиск ключа, не являющегося точкой сплайна.*

В этом случае возможен вариант, когда искомый ключ меньше ключа точки сплайна, на которую указывает элемент *RadixTable*. Тогда, следуя алгоритму авторов, поиск начнется с первой точки сплайна, содержащей ключ, больший искомого ключа. И ключ не будет найден.

В тестовой реализации поиск *ключа, не являющегося точкой сплайна*, осуществляется не от первого сплайн-поинта данного префикса, а от предыдущего. В этом случае искомый ключ будет точно находиться в искомом интервале.

3. *Корректное заполнение RadixTable при условии отсутствия точки сплайна (сплайн-поинт) с необходимым префиксом*.

Так как префиксы ключей реальных данных не всегда образуют множество всех битовых комбинаций, часть элементов RadixTable могут быть пустыми, т. е. не будут содержать указателей на точки сплайна.

Для стандартизации поиска в п. 2, если элемент RadixTable с номером $i + 1$ оказался пустым, заполняем его значением элемента i.

4. *Уточнение бит префикса при заполнении RadixTable*.

Входные данных могут иметь множество ключей с одинаковым набором начальных бит.

Например, если набор данных начинается с ключа 110000, а заканчивается 111111, то при $r = 2$ префикс у всех ключей будет одинаковый. RadixTable во всех ячейках будет содержать одну ссылку и не будет выполнять роль индекса уменьшающего объем поиска. Увеличение r может улучшить ситуацию, но возрастет память под RadixTable. Сохраняя размер префикса, можно убрать одинаковые начальные биты у всех ключей (в примере – “11”), и заполнять RadixTable префиксами со следующими r-битами. Таким образом можно добиться более эффективного использования RadixTable.

Эксперименты

Эксперименты проводились посредством запусков программы на датасете MovieLens32M [4]. Размер входного файла ограничен 25 Мб для согласования с размером SST файла RocksDB [5]. Время поиска – это суммарное время поиска каждого ключа из датасета.

1. Исследование зависимости времени поиска при одновременном изменении погрешности (ошибки) сплайна и количества бит префикса.

График зависимости показан на рис. 2. Значения ошибки начинаются с 2.

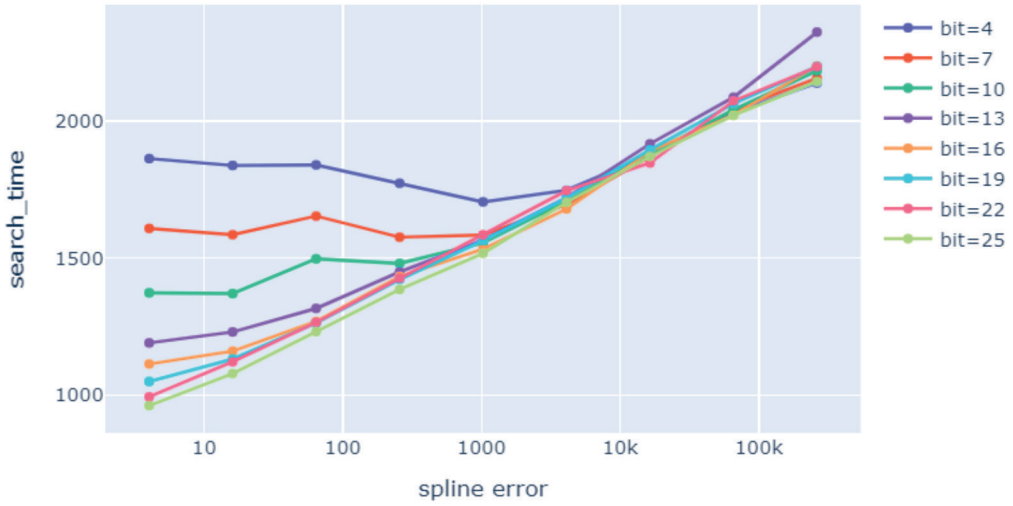

Рис. 2. Зависимость времени поиска от ошибки сплайна и бит префикса
Fig. 2. Dependence of search time on spline error and prefix bit

Можно обратить внимание, что время поиска при определенных ошибках меньше зависит от количества бит префикса. Это объясняется тем, что при увеличении ошибки количество точек сплайна GreedySplineCorridor становится меньше, чем размер RadixTable, и дальнейшее увеличение RadixTable не имеет смысла.

2. Исследование зависимости количества сплайн-точек от величины ошибки сплайна.

Так как GreedySplineCorridor никак не зависит от RadixSpline, эксперимент проводится при фиксированном количестве бит префикса. График представлен на рис. 3. Эксперимент прекращается при аппроксимации всех данных двумя точками сплайна, так как дальнейшее увеличение ошибки не имеет смысла.

Рис. 3. Зависимость количества точек сплайна от величины ошибки сплайна
Fig. 3. Dependence of the number of spline points on the spline error value

3. Исследование зависимости количества точек сплайна и времени поиска от ошибки сплайна и бит префикса.

График зависимости количества точек сплайна и времени поиска от двух параметров: ошибки сплайна и бит поиска представлен на рис. 4. На графике каждая точка является новым radix-spline, с уникальным набором параметров. Каждый график отображает результат экспериментов поиска с фиксированным количеством бит префикса и меняющимся значением ошибки. По каждому созданному сплайну измерено время поиска всех элементов исходных данных и зафиксировано количество сплайн-точек. Для сравнения также приведено время простого бинарного поиска всех элементов (прямая BinSearch).

Данный эксперимент показывает, что уже начиная с префикса в 10 бит, можно получить значительный выигрыш по времени относительно простого бинарного поиска. А при использовании большого количества памяти можно проводить поиск быстрее на треть (1000 мс относительно 1500 мс).

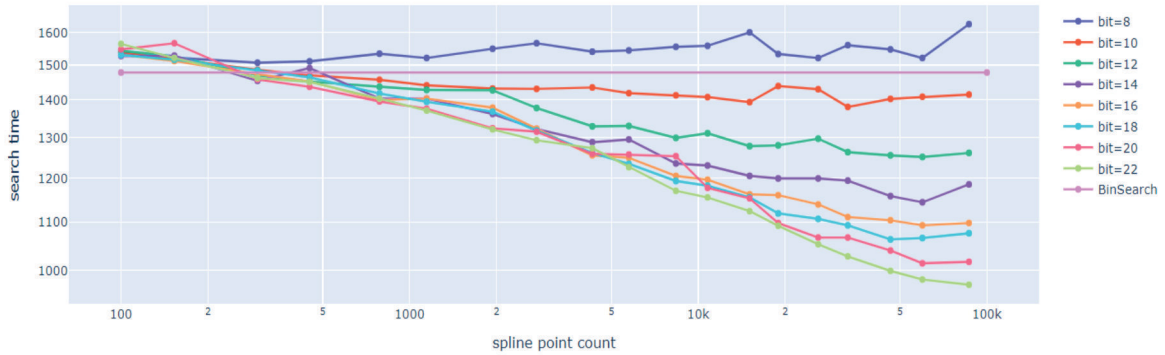

Рис. 4. Зависимость количества точек сплайна и времени поиска от ошибки сплайна и бит префиска
Fig. 4. Dependence of the number of spline points and search time on the spline error and prefix bit

4. Исследование зависимости используемой памяти и времени поиска от ошибки сплайна и бит префикса.

Используемую RadixSpline память можно рассчитать из размера RadixTable, сохраняющей 2^r ссылок на элементы, и памяти, занимаемой точками GreedySplineCorridor.

В связи с планами заменить разреженный индекс Index Block Format RocksDB [5] на RadixSpline вычислен и «эталон» используемой памяти. В Index Block Format RocksDB все данные разделены на блоки, и для каждого блока в структуру сохраняется ключ с максимальным значением. Для исходного набора данных рассчитано количество необходимых блоков, сохранены ключи со средним интервалом и рассчитана память для сохранения этих ключей.

Результат эксперимента приведен на рис. 5. Здесь можно заметить, как сильно растет используемая память при увеличении количества бит префикса. В случае неограниченной памяти выигрыш можно получить значительный, но неограниченной памяти в реальных проектах не существует.

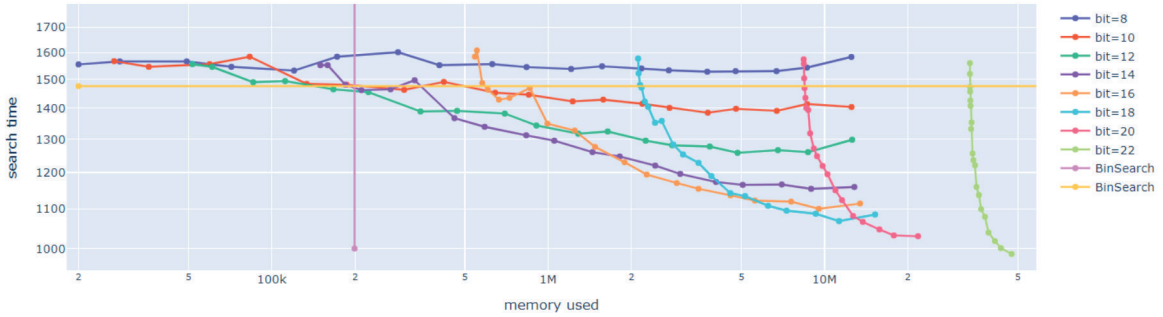

Рис. 5. Зависимость используемой памяти и времени поиска от ошибки сплайна и бит поиска
Fig. 5. Dependence of memory used and search time on spline error and search bits

5. Поиск оптимального соотношения параметров.

Выделим часть графика на рис. 5, где RadixSpline работает быстрее, чем бинарный поиск, но при этом требует меньше памяти, чем подсчитанная ранее структура разреженного индекса Index Block Format RocksDB. График представлен на рис. 6.

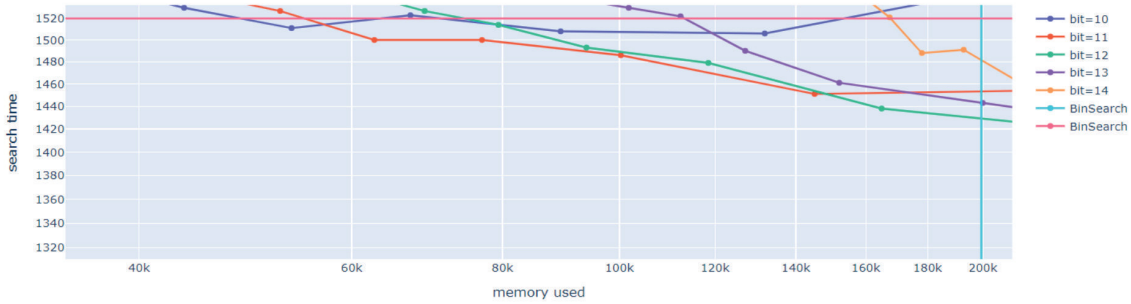

Рис. 6. Поиск оптимальных параметров
Fig. 6. Search for optimal parameters

Даже в этом интервале есть различные наборы параметров, которые будут лучше в различных ситуациях. Если не превышать затраты по памяти, можно выиграть 5 % во времени поиска или сэкономить более 80 % памяти, не проигрывая по времени. Следовательно, есть смысл продолжать исследовать данную структуру и встроить в рабочий продукт.

Заключение

В работе проведен анализ и тестирование обученного индекса RadixSpline, представляющего собой комбинацию сплайн-аппроксимации с заранее известной погрешностью и раз-

реженного индекса. Тестовая реализация RadixSpline позволила убедиться, что это гибкий и эффективный инструмент для индексации и поиска в отсортированных данных. Он довольно простой, не требует нескольких проходов по данным для построения, а также может быть полезен в потоковой обработке.

Результаты экспериментов показали, что RadixSpline способен значительно увеличить скорость поиска, в сравнении с бинарным поиском, а также может занимать меньше памяти, чем нынешний разреженный индекс в RocksDB. При этом, меняя параметры структуры, возможно настроить баланс скорости и объем памяти конкретно под каждую задачу.

Исследование параметров тестовой реализации RadixSpline показывает, что она может быть использована в промышленных продуктах. В настоящее время она находится на стадии внедрения в RocksDB. Кроме того, для улучшения производительности реализации рассматриваются варианты другого метода сплайн-аппроксимации.

Таким образом можно сказать, что RadixSpline является перспективным и интересным инструментом для исследования и реализации и может стать началом важной ветки развития обучаемых индексов.

Список литературы / References

1. **Kraska T., Beutel A., Chi E.H., Dean J., Polyzotis N.** The Case for Learned Index Structures // arXiv:1712.01208 [cs.DB]. 2017. DOI: 10.48550/arXiv.1712.01208
2. **Kipf A., Marcus R., van Renen A., Stoian M., Kemper A., Kraska T., Neumann T.** RadixSpline: A Single-Pass Learned Index // Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM ‘20). 2020. Art. No. 5. P. 1–5. DOI: 10.1145/3401071.3401659.
3. **Neumann T., Michel S.** Smooth Interpolating Histograms with Error Guarantees // Proceedings of the 11th International Conference on Extending Database Technology (EDBT ‘08). Cardiff, UK, 2008. P. 42–53. DOI: 10.1007/978-3-540-70504-8_12.
4. MovieLens32M Dataset [Электронный ресурс] / URL: GroupLens Research (дата обращения: 10.12.2024).
5. Index Block Format [Электронный ресурс] // RocksDB Wiki. – URL: https://github.com/facebook/rocksdb/wiki/Index-Block-Format (дата обращения: 10.08.2024).
6. RocksDB: A Persistent Key-Value Store [Электронный ресурс] / Meta Open Source. URL: https://github.com/facebook/rocksdb (дата обращения: 10.08.2024).
7. O’Neil P. The Log-Structured Merge-Tree (LSM-Tree) // Acta Informatica. 1996. Vol. 33. No. 4. P. 351–385. DOI: 10.1007/s002360050048.
8. **Marcus R., Kipf A., van Renen A., Stoian M., Misra S., Kemper A., Neumann T., Kraska T.** Benchmarking Learned Indexes // Proceedings of the VLDB Endowment (PVLDB). 2021. Vol. 14. No. 1. P. 1–13. DOI: 10.14778/3421424.3421425.
9. **Li Z., Chan T. N., Yiu M. L., Jensen C. S.** A Survey of Learned Indexes for the Multi-dimensional Space // arXiv:2403.06456 [cs.DB]. 2024. DOI: 10.48550/arXiv.2403.06456
10. **Ferragina P., Vinciguerra G.** A Critical Analysis of Recursive Model Indexes // arXiv:2106.16166 [cs.DB]. 2021. DOI: 10.48550/arXiv.2106.16166
11. **Ding J., Minhas U. F., Yu J., Wang C., Li Y., Zhang H., Chandramouli B., Gehrke J., Kossmann D., Lomet D., Kraska T.** Towards Practical Learned Indexing (Extended Abstracts) // arXiv:2108.05117 [cs.DB]. 2021. DOI: 10.48550/arXiv.2108.05117
12. **Wu J., Zhang Y., Chen S., Wang J., Chen Y., Xing C.** LSI: A Learned Secondary Index Structure // arXiv:2205.05769 [cs.DB]. 2022. DOI: 10.48550/arXiv.2205.05769
13. **Hadian A., Heinis T.** Shift-Table: A Low-Latency Learned Index for Range Queries using Model Correction // arXiv:2103.13160 [cs.DB]. 2021. DOI: 10.48550/arXiv.2103.13160

14. **Vaidya K., Kraska T.** Partitioned Learned Bloom Filter // Proceedings of the VLDB Endowment. 2020. Vol. 13. No. 12. P. 3298–3311. DOI: 10.14778/3415478.3415545
15. **Spector B., Marcus R., van Renen A., Stoian M., Kemper A., Kraska T., Neumann T.** Bounding the Last Mile: Efficient Learned String Indexing // arXiv:2111.14905 [cs.DB]. 2021. DOI: 10.48550/arXiv.2111.14905
16. **Crotty A.** Hist-Tree: Those Who Ignore It Are Doomed to Learn [Электронный ресурс] // CIDR 2021. URL: https://cs.brown.edu/people/acrotty/pubs/cidr2021_paper20.pdf (дата обращения: 02.02.2025).

Информация об авторах

Стецкий Илья Максимович, магистрант

Пищик Борис Николаевич, доцент
SPIN-код: 4297-3046
AuthorID: 13080

Information about the Authors

Ilya M. Stetsky, Master's Student
Boris N. Pishik, Associate Professor
SPIN-cod: 4297-3046
AuthorID: 13080

Статья поступила в редакцию 14.04.2025;
одобрена после рецензирования 20.05.2025; принята к публикации 20.05.2025

The article was submitted 14.04.2025;
approved after reviewing 20.05.2025; accepted for publication 20.05.2025