Попытка создания равномерного индексирования на части столбцов

А.И. Миронов

***Аннотация*— Из-за роста объема данных и разнообразия требований к их обработке, все чаще переходят от обработки данных в реальном времени к использованию предварительно сохраненных и подготовленных результатов. В ряде случаев СУБД решают проблемы производительности за счет увеличения использования памяти, однако важно рассмотреть варианты экономии памяти при сохранении результатов методов, использующих такие подходы, как индексирование, хеширование и нейросетевые алгоритмы.**

В статье рассматривается метод повышения эффективности поисковых запросов в СУБД для таблиц большого размера. Предлагаемый метод основан на индексировании с возможностью поиска по части индексируемых столбцов. В его основе лежат элементы кластеризации на основе карты Кохонена, а также сохранение дополнительных метаданных. Такой подход мог бы позволить избавится от сложного вложенного индексирования в сравнении с классическим b-tree. Правильное применение данного подхода может позволить эффективно обрабатывать таблицы, имеющие разные поисковые потребности, по разным группам столбцов, для которых хранение индексации для каждого большого типа запроса или группы запросов может приводить к значительным затратам памяти, а также потере производительности при работе с большими блоками памяти, рост которых не является линейным.

***Ключевые слова*— индексирование, параллельное программирование, BigData, поиск сложных структур, кластеризация, базы данных, распределенные вычисления, карты Кохонена.**

I. Введение

За последние годы объем мультимедийной информации значительно вырос, увеличившись в несколько раз. Согласно различным оценкам, общий объем данных, хранящихся в Интернете и на физических носителях, уже превышает 170 зеттабайт. Одновременно с этим заметно усложняются структуры данных, которые необходимо обрабатывать. Это особенно актуально с учетом того, что современные портативные устройства становятся мощнее, позволяя пользователям работать с гораздо более сложными информационными структурами. В таких условиях задача поиска остается одной из ключевых, однако её реализация становится все более сложной из-за изменений в характере данных и запросов. Современные поисковые запросы уже не ограничиваются простыми операциями по нахождению одного элемента среди множества; они стали гораздо более сложными и неопределенными по своей структуре.

Во-первых, многие поисковые запросы сейчас имеют неточный характер и не нацелены на получение конкретного совпадения. Все больше запросов работают с диапазонами значений, нацеленными на извлечение наиболее релевантной информации, а не на точное совпадение. Во-вторых, сами данные, с которыми работают современные системы, становятся более многослойными и сложными по своей структуре. Системы управления базами данных (СУБД) и поисковые механизмы вынуждены адаптироваться к поисковым моделям, которые могут сильно отличаться от изначально ожидаемых, но при этом соответствовать заданным критериям. Однако существующие поисковые алгоритмы и системы не всегда могут полноценно справиться с новыми задачами, которые предъявляют высокие требования к скорости и точности обработки данных. Это происходит главным образом потому, что подходы, предназначенные для работы с простыми структурами данных, часто не меняются в ответ на более обобщенные задачи обработки данных. Несмотря на то, что характер обработки информации стал более универсальным и многозадачным, многие алгоритмы остаются зафиксированными на работе с базовыми типами данных, что приводит к несовпадению возможностей используемых инструментов с текущими требованиями и архитектурными изменениями.

В свете этих изменений становится необходимым разрабатывать новые алгоритмы, которые могли бы выполнять базовые операции, такие как обработка поисковых запросов, с учётом новых требований и обобщенных подходов к обработке данных, при этом обеспечивая приемлемый уровень производительности. Внедрение таких усовершенствованных алгоритмов, на которых строится значительная часть современных программных решений (включая поисковые механизмы), позволит достичь значительного прироста в производительности. Чем сложнее и многоуровневее будет программное обеспечение, тем заметнее окажется преимущество использования новых алгоритмов перед традиционными методами. Это особенно важно в условиях текущей тенденции к усложнению программных и аппаратных комплексов, где производительность и эффективность становятся ключевыми параметрами.

В данной статье рассматривается возможность предложения метода, основанного на снижении размерности за счет карт Кохонена с применением частичной индексации для оптимизации задачи

Статья получена 7 июля 2025.

А.И. Миронов – Смоленский государственный университет (email: 615153@mail.ru)

поисковых запросов по всем/нескольким столбцам.

II. Цель исследования

В настоящее время данные, обрабатываемые внутри СУБД, обладают сложной структурой по составу столбцов и их взаимосвязям. Ситуации, когда работа ведется исключительно с простыми числовыми значениями, становятся редкостью. Всё чаще данные, представленные в табличной форме, представляют собой сложные объекты, описываемые множеством различных характеристик, которые формируют длинные вектора. Поиск по большим массивам данных, особенно когда необходимо осуществить частичный поиск по множеству характеристик, которым должны соответствовать значения искомого объекта, становится привычной задачей. Тем не менее, такие задачи зачастую решаются с помощью алгоритмов, изначально разработанных для работы с единичными значениями, что нередко приводит к снижению производительности [1, 2].

Целью данного исследования является попытка создать такой алгоритм, который при помощи необычного индексирования рассчитан на частые неоднородные по сути поисковые запросы к большим таблицам с расчет на параллельную обработку [3]. Данный алгоритм для кластеризации множеств предполагает способ, основанный на упрощенном виде карт Кохонена, а также мер близости на пространстве элементов таблицы с целью осуществлять поиск по заранее созданной структуре данных без ее изменения с любым поисковым образом в отличии от классической индексации.

III. Кластеризация на основе карт Кохонена

Первая составляющая предлагаемого алгоритма – нахождение метаданных для будущего индексирования. Для этого нужно использовать упрощенный вариант карт Кохонена: слой Кохонена состоит из n параллельно работающих линейных элементов, каждый из которых имеет m координат, соответствующих количеству столбцов. Все линейные элементы сравниваются по парам с центрами кластеров. После этого выбирается центр, который находится ближе всего к обучающему примеру, для дальнейшей модификации. Если несколько центров имеют максимальное сходство, то изменяются либо все такие центры, либо только первый из них (в зависимости от установленного соглашения). Количество кластерных центров соответствует числу кластеров, к которым изначально распределяются обучающие примеры, и затем происходит их перераспределение. Число входных переменных равно количеству столбцов, характеризующих элементы, на основании которых объекты классифицируются в соответствующие кластеры. После того как введены основные понятия, можно описать общую схему "обучения" кластерной карты Кохонена. Данная схема относится к методам машинного обучения, хотя она представляет собой достаточно примитивный элемент этой области, где линейные центры можно трактовать как нейроны. Карты Кохонена широко применяются для решения задач кластеризации, и для их корректного функционирования требуется учитывать определенные их характеристики, что также рассматривается в данной работе.

Алгоритм нахождения центров включает следующие этапы:

1. Задание структуры “сети” (количества центров слоя глубины) (M);
2. Случайная инициализация весовых коэффициентов центров;
3. Подача случайного элемента текущего обучения и расчет расстояний по выбранной метрики или меры близости на базе таблицы, от входного вектора до центров всех будущих кластеров:
$$R_j = \sum \left| x_i - w_i \right|$$
4. По наименьшему из значений Rj будет выбран центр-победитель j, в наибольшей степени близкий по значениям с входным элементом. Для выбранного центра (и только для него) выполняется коррекция весовых коэффициентов – его сдвиг:
$$w_{i,j}\left(q+1\right) = w_{i,j}\left(q\right) + v\left(x_i - w_{i,j}(q)\right)$$
где v – коэффициент скорости обучения;
5. Цикл повторяется с шага 3 до выполнения одного или нескольких условий окончания: исчерпано заданное предельное количество эпох обучения; не произошло значимого изменения весовых коэффициентов в пределах заданной точности на протяжении последней эпохи обучения; исчерпано заданное предельное физическое время обучения[4].

Коэффициент скорости обучения может задаваться постоянным из пределов (0, 1] или переменным значением, постепенно уменьшающимся от эпохи к эпохе в данном рассмотрении, он полагается именно постоянным, а эпохи всего две что делает обучение тривиальным и достаточно быстрым.

После такой процедуры каждый элемент множества будет сравнен с каждым центром и значит косвенно с каждым другим элементом на каждом уровне глубины. Номер того центра, что является ближайшим к входному объекту и является номером кластера, который соответствует входному элементу. [8] Центры кластера – случайные и сдвигаемые элементы которые являлись кластеризирующими в процессе. Учитывая тот факт, что эти объекты имеют ту же структуру что и объекты кластеризируемого множества, можно считать, что они так же принадлежат соответствующему кластеру – стоят в центре. Так же можно видеть, что кластеризация происходит при одних и тех же начальных весах одинаково. То есть если имеются начальные значения:
$$\left\{\left\{x_1, x_2, \ldots, x_k\right\}_{\ldots}, \left\{x_1, x_2, \ldots, x_k\right\}\right\}$$
то кластеризация одного и того же множества при том же порядке элементов пройдёт таким же образом и с тем же результатом[5,6,7].

IV. Предлагаемый способ индексирования

В основе индексирования будут положены принципы

сосредоточения на отсечении ненадлежащих (заведомо отдаленных по мерам схожести) имеющихся объектов внутри базы данных, а не на схождении к похожим образам. Большинство современных индексов по нескольким столбцам строятся композиционно: сначала выполняется индексирование по одному полю, затем — по следующему и так далее. Однако такая структура сильно усложняет добавление новых данных, поскольку требует множества операций сравнения. Более того, при изменении ключей для поиска, например, исключении одного столбца, эффективность индекса существенно снижается. Такой способ организации также негативно сказывается на использовании памяти — если данные сильно похожи и деревья имеют большую глубину, объем памяти, необходимой для хранения индексов, значительно возрастает. Дополнительно это усложняет ситуацию при работе с журналами транзакций, требующими еще больше пространства.

Подобную картину возможно исправить, необходимо отойти от вложенности сравнений при выстраивании деревьев поиска или видоизменить концепцию деревьев, сократив влияние пересечения конкретных значений столбцов. При такой реализации кортеж столбцов для поиска будет не важен с точки зрения производительности. Дальнейший алгоритм будет представлен на концептуальном примере для простоты изложения. Предположим, что у нас существует таблицы с достаточным количеством столбцов (очевидно, что для малого их числа достаточно стандартных индексов) и немалым количеством строк, для целей оптимизации поиска и сравнения.

Представим элементы этой таблицы как элементы гиперплоскости, а каждый столбец – их координатами. Для вычисления расстояния выберем на этой гиперплоскости метрику Таксиста, она более чем другие удовлетворяет нашим целям, поскольку не “сглаживает” разницу в нескольких конкретных координатах в угоду общей разницы[8]. Так же определим понятие “частичного элемента” – это элемент, всех координат которого мы не знаем, нам известны лишь какие-то k его координат, причем не обязательно первые k но для простоты изложения будем полагать их. Определим “частичное расстояние” между частичным элементом и стандартным элементом гиперплоскости. В нашем случае, расстоянием назовем расстояние между частичным элементом, и элементом, состоящим из тех же координат целевого элемента, которые известны о частичном. Например если имеется элемент (1, 2, 3) и (?, ?, 2) то расстоянием между ними будем считать 1.

$$|0|+|0|+|3-2|=1$$

Основываясь на описанных выше определениях, начнем разбивать наше множество (на первом шаге всю таблицу) на подмножества путем использования схемы схожей с картами Кохонена. А именно выберем M элементов со случайными координатами, случайно лежащих среди элементов нашей гиперплоскости. Затем вычислим расстояние между каждым элементом таблицы и этими вершинами, при обработке каждого элемента, будем присуждать этому элементу ту вершину, которая наиболее близка к нему на данном шаге и “сдвигать” с некоторым коэффициентом выбранную вершину к обрабатываемому элементу.

Процесс можно повторять неоднократно, но это лишь немного влияет на эффективность конечной модели. После подобного передвижения назовем центрами эти вершины, определим каждый элемент таблицы к ближайшему центру, при этом для центра сохраним два значения “Максимальный радиус” - R и “Минимальный радиус” - r. Первое - максимальное расстояние от центра до любого принадлежащего центру элемента. Второе – минимальное расстояние от центра до любого элемента по одной координате, то есть минимум разности любой координаты центра с этой же координатой любого его элемента.

После, будем повторять процедуру до тех пор, пока в результирующих множествах разбиения не окажется достаточно мало элементов (максимум до M дальнейшее разбиение не выгодно, поскольку глубина разбиения будет в меньшей степени влиять на ширину). Отметим, что для одинаковых элементов гарантированно попадание к одним центрам, поскольку если два элемента равны между собой, они одинаково близки к другим элементам и потому два равных элемента будут отнесены к одному центру. На данном этапе имеется система, которая уже способна работать как индекс для поиска по всем столбцам таблицы. При поиске элемента достаточно проделать с ним одним операцию “примерки”. А именно сравнивать его с центрами на первой “глубине” и отнести к одному из них. Затем проделать ту же процедуру на данной глубине и так далее и к все более глубоким, наиболее близким ему центрами. В конечном итоге такого сравнения поисковый образ попадет в то множества, где точно находятся/находились бы элементы эквивалентные ему. Это просто объяснить поскольку процесс попадания элементов для хранение схож, а как видно для схожим элементов получаем одни результаты. И уже среди этих элементов производить сравнение. Понять, что в таком случае скорость поиска будет склонятся с логарифмической не составляет труда– поскольку представляет из себя проход по одной ветви дерева.

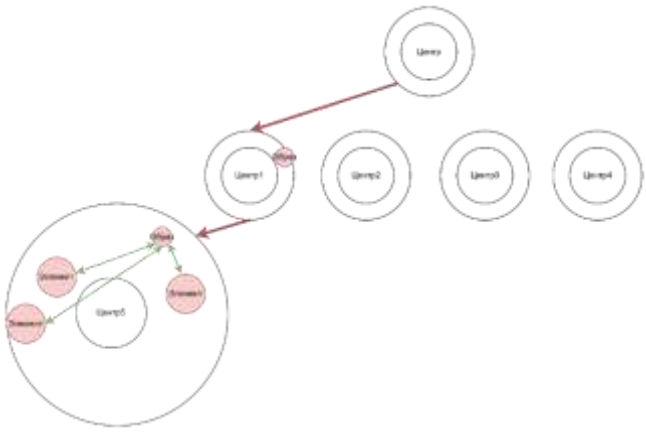

Рис. 1: Поиск

Следует отметить, что разбиение производится только по тем столбцам, которые участвуют в сравнении, так как уникальные поля не влияют на результат и будут

восприниматься как случайный шум при расчете расстояний. Хотя технически возможно выполнить разбиение по всем столбцам, это не окажет негативного влияния на вычисления и поиск. Такой индекс обеспечивает эффективный поиск элементов по нескольким столбцам, причем порядок столбцов не имеет значения, что является важным отличием от классических индексов. Также стоит подчеркнуть, что перебалансировка подобной структуры значительно проще, так как алгоритм расчета каждого уровня линейный, и нет необходимости строить вложенные деревья для каждого столбца.

V. Поиск на основе индексирования

Перейдем к вопросу о поиске образов по набору из части столбцов участвующих в таком совместном индексе. Для того чтобы по данному на вход поисковому образу определить множество для сравнения, необходимо находить ближайшие к нему центры по частичному расстоянию, а далее, не переходить к вложенным центрам, а находить описанные ваше, максимальное и минимальное частичные расстояния от выбранного центра. В случае если хранящееся для центра максимальный радиус меньше, чем расстояние от поискового образа – поиск можно прекращать, поскольку ни один из элементов базы не будет идентичен данному, что можно доказать следующим образом:

Пусть R1 и r1 соответственно большой и малый радиусы, описанные выше, центра T1, а C - поисковый образ с неполным набором столбцов, иными словами

последствия

$$\forall a : a = (a_1, a_2, ..., a_n) \in T_1 : \rho(a, T_1) < R,\ \forall j \left| a_i - t_j \right| > m$$

Тогда если

$$\hat{\rho}(C, T_1) > R_1,$$

- частичное расстояние.

То среди элементов относящихся к любому из вложенных кластеров в T1 нет ни одного совпадающего с C, поскольку любой элемент уже по полному расстоянию не превосходит R, а в данном случае расстояние может только расти.

В случае, если максимум расстояний, хранящийся для центра, окажется большим следует сравнить минимум частичного расстояния хранящегося для центра, с минимумом частичного расстояния для данного поискового образа, в том случае, если первое окажется больше, следует перейти к сравнению со следующим центром, поскольку ни один элемент не будет эквивалентен около данного центра поисковому образу.

Однако в рассуждения можно пойти и дальше. Для целей поиска ни одна координата не должна быть меньше ri. Видно, что расстояние по любой из координат для центра Ti не превосходит ri. Аналогично расстояние общее не может превосходить R1. А если так, то мы можем положить что:

$$\hat{\rho}(C, T_1) \le \rho(C, T_1) \le R_1$$

И далее

$$R_1 \ge \hat{\rho}(C, T_1) + k \cdot r_1,$$

k - число неизвестных столбцов поискового элемента.

И далее

$$\hat{\rho}(C, T_1) \ge t \cdot r_i,$$

t - количество известных столбцов.

Таким образом получим в совокупности для всех центров

$$R_{ij} - k \cdot r_{ij} \ge \hat{\rho}(C, T_{ij}) \ge t \cdot r_{ij},$$

i - номер центра, j-номер уровня

На каждом уровне вложенности необходимо проводить аналогичную проверку, которая выполняется достаточно просто, так как вычисление расстояния сводится к сравнению элементов, особенно если используется манхэттенское расстояние, а остальные компоненты хранятся как постоянные значения. Однако, если проверка не проходит, сразу отсекается целый кластер элементов, которые заведомо не подходят для совпадения с поисковым образом.

По мере прохождения через каждый центр до его вложенных под-центров, поисковый образ сравнивается с множеством элементов, которые потенциально могут быть близки к нему, так как они находятся вблизи того же вложенного центра, что и элементы базы данных. При этом элементы, отсеченные ранее как явно не соответствующие поисковому образу, в дальнейших сравнениях не участвуют. Этот подход отличается от традиционного индексного метода, где цель индекса — указать путь к множеству потенциально равных элементов. В данном же подходе основное внимание уделяется отсечению заведомо не равных элементов.

Используя данный метод, можно с помощью одной структуры определять схожие элементы, искать строки по полной схеме или по части столбцов. Каждый такой поиск выполняется быстрее, чем полный перебор, при этом значительно экономится дисковое пространство и уменьшаются затраты на метаданные. Важно отметить, что данный подход не теряет эффективности при добавлении новых столбцов, так как можно в фоновом режиме пересчитывать кластеры. Кроме того, он универсален для поиска по различным наборам столбцов и, чем больше столбцов используется, тем быстрее идет процесс поиска за счет уменьшения выборки. В то же время, чем меньше столбцов задействовано, тем проще становится вычисление условий.

Может показаться что процедура разбиения листового множества велика, однако нужно помнить, что элементов в листовых множествах мало и потому его разбиение не представляется трудным и в некоторых системах может осуществляться что называется “на лету”.

VI. Заключение

Сегодня становится все более очевидной необходимость замены множества простых алгоритмов, которые не обеспечивают максимальную производительность, на новые подходы, способные повысить эффективность и расширить возможности в

различных областях. Разнообразие задач, стоящих перед современным программным обеспечением и возникающих в разных сферах, требует разработки и внедрения инновационных методов и принципов. Важной задачей становится не только создание фундаментальных решений, но и эффективная обработка частных случаев, поскольку даже они, учитывая огромные объемы данных, играют значительную роль в процессе обработки информации. С другой стороны, важно также укреплять основные подходы к организации обработки данных, так как уже нельзя полагаться исключительно на производительность оборудования для компенсации недостатков в алгоритмах обработки данных.

Таким образом, становится ясно, что текущая ситуация требует поиска не просто локальных улучшений, а принципиально новых методов обработки данных и решения программных задач с тщательным учетом всех нюансов и особенностей этих задач. Особенно заметен этот тренд в задачах, которые традиционно считались простыми и не требовали разработки новых методов. Однако, по мере увеличения масштабов, такие задачи могут стать узким местом, существенно снижающим производительность. Примером может служить операция поиска. Как было показано ранее, одним из перспективных подходов к обработке поисковых запросов может быть активное использование метаданных в алгоритмах, что открывает новые возможности для улучшения работы:

- Увеличение способности к поиску по общему числу столбцов без лишних затрат;
- Переложение значительной части вычислительных затрат на затраты памяти;
- Освобождение снижения требования на типы данных для осуществления поиска;
- Решение некоторых технических проблем параллелизма;
- Оптимизацию многих поисковых запросов путем непосредственного сравнения только в заключении;
- Исключение линейного роста сложности с ростом данных;
- Обработку некоторых задач прежде, чем те будут поставлены – имеется в виду оценка количества.

Исходя из описанного выше, становится очевидным, что алгоритмы, основанные на интенсивном использовании различных видов метаданных, могут стать перспективной альтернативой классическим, устаревающим методам решения задач. Примером такого решения является предложенная схема выполнения поисковых запросов, которая в значительной степени опирается на метаданные базы данных. В данной работе была представлена схема вышеописанного алгоритма. Подобные предлагаемой схеме решения, уже включаются постепенно в жизненные циклы многих программно-аппаратных комплексов, однако необходимо усиливать участие подобных алгоритмов в разработке, что позволит существенно повысить скорость и эффективных всех этапов разработки данных, поскольку представленные проблемы находятся у истоков большинства современных алгоритмов обработки данных.

Библиография

[1] Abdel-Basset M. et al. An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems //Personal and Ubiquitous Computing. – 2018. – Т. 22. – №. 5-6. – С. 1117-1132.

[2] Chamoso, Pablo, et al. "Social computing for image matching." PloS one 13.5 (2018): e0197576.

[3] Das S. et al. Automatically indexing millions of databases in microsoft azure sql database //Proceedings of the 2019 International Conference on Management of Data. – 2019. – С. 666-679.

[4] Dodonov A. et al. Method of Parallel Information Object Search in Unified Information Spaces //International Journal of Computer Network and Information Security (IJCNIS). – 2021. – Т. 13. – №. 4. – С. 1-13.

[5] Gorokhovatskyi V. A., Gorokhovatskiy A. V., Peredrii Y. O. Hashing of structural descriptions at building of the class image descriptor, computing of relevance and classification of the visual objects //Telecommunications and Radio Engineering. – 2018. – Т. 77. – №. 13.

[6] Graefe G. et al. Modern B-tree techniques //Foundations and Trends® in Databases. – 2011. – Т. 3. – №. 4. – С. 203-402.

[7] Haynes, David, et al. "High performance analysis of big spatial data." 2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015.

[8] Iljin P. L., Munerman V. J. Recursive computation of the multidimensional matrix determinant. Systems of computerized mathematics and their appendices: XX International Scientific Conference. Smolensk: SmolSU publishing. 2019. Vol. 1, Issue 20. pp. 162-166. (In Russ).

[9] Kirikova A., Mironov A. Using Metadata-indexing to Improve the Efficiency of Complex Operations //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). – IEEE, 2021. – С. 2124-2127.

[10] Kirikova A., Mironov A., Munerman V. The Method of Composition Hash-functions for Optimize a Task of Searching Images in Dataset //2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2020. – С. 1983-1986.

[11] Levin N. A., Munerman V. I. Models of big data processing in massively parallel systems //Системы высокой доступности. – 2013. – Т. 9. – №. 1. – С. 035-043.

[12] Lomet D. The evolution of effective b-tree: Page organization and techniques: A personal account //ACM SIGMOD Record. – 2001. – Т. 30. – №. 3. – С. 64-69.

[13] Lvovich I. et al. Modeling and optimization of processing large data arrays in information systems //2021 International Conference on Information Technology and Nanotechnology (ITNT). – IEEE, 2021. – С. 1-5.

[14] Monga, Vishal, and Brian L. Evans. "Perceptual image hashing via feature points: performance evaluation and tradeoffs." IEEE Transactions on Image Processing 15.11 (2006): 3452-3465.

[15] Munerman V., Munerman D. Realization of Distributed Data Processing on the Basis of Container Technology //2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2019. – С. 1740-1744.

[16] Munerman V., Munerman D., Samoilova T. The Heuristic Algorithm For Symmetric Horizontal Data Distribution //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). – IEEE, 2021. – С. 2161-2165.

[17] Munerman V.I. The experience of massive data processing in the cloud using windows azure (as an example) High availability systems. - 2014. - V. 10. - №. 2. - p. 8-13.

[18] Pushpa Rani Suri, Sudesh Rani, "A New Classification for Architecture of Parallel Databases", Information Technology Journal, vol. 7, pp. 983. (2008).

[19] Pyurova T. A., Skvortsov S. V. CUDA technology and parallel computing On GPU //Informatics and applied mathimatic: interuniversity compendium of treatises, no. 21, pp. 163-166. 2015. (In Russ).

[20] Sridhar R. et al. Optimization of heterogeneous Bin packing using adaptive genetic algorithm //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2017. – Т. 183. – №. 1. – С. 012026. 16.

[21] Syrotkina O. et al. Mathematical Methods for optimizing Big Data Processing //2020 10th International Conference on Advanced

Computer Information Technologies (ACIT). – IEEE, 2020. – C. 170-176.
[22] Wajszczyk B., Gruszka I. M. Analysis of possibilities to increase the efficiency of the relative database management system using the methods of parallel processing //Radioelectronic Systems Conference 2019. – SPIE, 2020. – T. 11442. – C. 385-398.
[23] Zakharov V. et al. Architecture of Software-Hardware Complex for Searching Images in Database //2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2019. – C. 1735-1739.
[24] Zakharov V. N., Muneman V. I., Samoilova T. A. Parallel methods for deriving associative rules with the usage indatabase and in-memory technologies //CEUR Workshop Proceedings. – 2017. – C. 219-225.
[25] Zobel J., Moffat A., Sacks-Davis R. An efficient indexing technique for full-text database systems //PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES. – INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS (IEEE), 1992. – C. 352-352.

Attempt to create uniform indexing on subsets of columns

Artem Mironov

***Abstract* - Due to the growing volume of data and the increasing variety of processing requirements, there is a shift from real-time data processing to the use of pre-saved and pre-prepared results. In some cases, DBMS address performance issues by increasing memory usage. However, it is important to explore memory-saving options while maintaining the results of methods based on approaches such as indexing, hashing, and neural algorithms.**

This article discusses a method for improving the efficiency of search queries in DBMS for large tables. The proposed method is based on indexing with the ability to search across specific indexed columns. It incorporates elements of clustering using a Kohonen map, as well as the preservation of additional metadata. This approach could help eliminate complex nested indexing compared to the classical B-tree. Proper implementation of this approach could allow efficient processing of tables with different search needs across various groups of columns, where maintaining indexing for each large query type or group of queries can lead to significant memory consumption and reduced performance when handling large memory blocks, the growth of which is not linear.

***Keywords* - indexing, parallel programming, BigData, complex structure search, clustering, databases, distributed computing, Kohonen maps**

REFERENCES

[1] Abdel-Basset M. et al. An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems //Personal and Ubiquitous Computing. – 2018. – Т. 22. – №. 5-6. – С. 1117-1132.

[2] Chamoso, Pablo, et al. "Social computing for image matching." PloS one 13.5 (2018): e0197576.

[3] Das S. et al. Automatically indexing millions of databases in microsoft azure sql database //Proceedings of the 2019 International Conference on Management of Data. – 2019. – С. 666-679.

[4] Dodonov A. et al. Method of Parallel Information Object Search in Unified Information Spaces //International Journal of Computer Network and Information Security (IJCNIS). – 2021. – Т. 13. – №. 4. – С. 1-13.

[5] Gorokhovatskyi V. A., Gorokhovatskiy A. V., Peredrii Y. O. Hashing of structural descriptions at building of the class image descriptor, computing of relevance and classification of the visual objects //Telecommunications and Radio Engineering. – 2018. – Т. 77. – №. 13.

[6] Graefe G. et al. Modern B-tree techniques //Foundations and Trends® in Databases. – 2011. – Т. 3. – №. 4. – С. 203-402.

[7] Haynes, David, et al. "High performance analysis of big spatial data." 2015 IEEE International Conference on Big Data (Big Data). IEEE, 2015.

[8] Iljin P. L., Munerman V. J. Recursive computation of the multidimensional matrix determinant. Systems of computerized mathematics and their appendices: XX International Scientific Conference. Smolensk: SmolSU publishing. 2019. Vol. 1, Issue 20. pp. 162-166. (In Russ).

[9] Kirikova A., Mironov A. Using Metadata-indexing to Improve the Efficiency of Complex Operations //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). – IEEE, 2021. – С. 2124-2127.

[10] Kirikova A., Mironov A., Munerman V. The Method of Composition Hash-functions for Optimize a Task of Searching Images in Dataset //2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2020. – С. 1983-1986.

[11] Levin N. A., Munerman V. I. Models of big data processing in massively parallel systems //Системы высокой доступности. – 2013. – Т. 9. – №. 1. – С. 035-043.

[12] Lomet D. The evolution of effective b-tree: Page organization and techniques: A personal account //ACM SIGMOD Record. – 2001. – Т. 30. – №. 3. – С. 64-69.

[13] Lvovich I. et al. Modeling and optimization of processing large data arrays in information systems //2021 International Conference on Information Technology and Nanotechnology (ITNT). – IEEE, 2021. – С. 1-5.

[14] Monga, Vishal, and Brian L. Evans. "Perceptual image hashing via feature points: performance evaluation and tradeoffs." IEEE Transactions on Image Processing 15.11 (2006): 3452-3465.

[15] Munerman V., Munerman D. Realization of Distributed Data Processing on the Basis of Container Technology //2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2019. – С. 1740-1744.

[16] Munerman V., Munerman D., Samoilova T. The Heuristic Algorithm For Symmetric Horizontal Data Distribution //2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). – IEEE, 2021. – С. 2161-2165.

[17] Munerman V.I. The experience of massive data processing in the cloud using windows azure (as an example) High availability systems. - 2014. - V. 10. - №. 2. - p. 8-13.

[18] Pushpa Rani Suri, Sudesh Rani, "A New Classification for Architecture of Parallel Databases", Information Technology Journal, vol. 7, pp. 983. (2008).

[19] Pyurova T. A., Skvortsov S. V. CUDA technology and parallel computing On GPU //Informatics and applied mathimatic: interuniversity compendium of treatises, no. 21, pp. 163-166. 2015. (In Russ).

[20] Sridhar R. et al. Optimization of heterogeneous Bin packing using adaptive genetic algorithm //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2017. – Т. 183. – №. 1. – С. 012026. 16.

[21] Syrotkina O. et al. Mathematical Methods for optimizing Big Data Processing //2020 10th International Conference on Advanced Computer Information Technologies (ACIT). – IEEE, 2020. – С. 170-176.

[22] Wajszczyk B., Gruszka I. M. Analysis of possibilities to increase the efficiency of the relative database management system using the methods of parallel processing //Radioelectronic Systems Conference 2019. – SPIE, 2020. – Т. 11442. – С. 385-398.

[23] Zakharov V. et al. Architecture of Software-Hardware Complex for Searching Images in Database //2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus). – IEEE, 2019. – С. 1735-1739.

[24] Zakharov V. N., Munerman V. I., Samoilova T. A. Parallel methods for deriving associative rules with the usage indatabase and in-memory technologies //CEUR Workshop Proceedings. – 2017. – С. 219-225.

[25] Zobel J., Moffat A., Sacks-Davis R. An efficient indexing technique for full-text database systems //PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES. – INSTITUTE OF ELECTRICAL & ELECTRONICS ENGINEERS (IEEE), 1992. – С. 352-352.