О структуре легко обновляемых полнотекстовых индексов

А. Б. Веретенников

alexander@veretennikov.ru

Кафедра вычислительной математики и компьютерных наук, УрФУ (Екатеринбург)

Аннотация

Рассматриваются стратегии организации обновляемых ассоциативных массивов во внешней памяти, применяемых для полнотекстового поиска. Изучаются вопросы создания индексов с разными видами ключа: одна форма слова, пара форм слов, последовательность форм слов. В зависимости от вида ключа, размер соответствующих ему данных различен, и структура их хранения может меняться. Приведены результаты экспериментов в контексте задачи полнотекстового поиска с учетом расстояния.

1 Введение

Ассоциативные массивы во внешней памяти применяются для полнотекстового поиска. В качестве ключа может выступать, например, слово, базовая форма слова, набор слов. Значением ключа является список записей, называемых словопозициями. Словопозиция – это запись вида (ID, P), где ID – идентификатор документа, а P – позиция ключа в документе, например, порядковый номер соответствующего слова в тексте документа. Такой ассоциативный массив называется инвертированным файлом или инвертированным индексом [1].

Будем решать задачу поиска с учетом расстояния (proximity search). Ищем документы, где слова искомого запроса располагаются близко друг к другу. Эта задача требует сохранения в индексе информации о каждом вхождении каждого ключа в каждом документе. Например, есть два файла 1.txt и 2.txt и в первом файле 10 раз встречается слово «мир», а во втором это слово встречается 5 раз, тогда мы имеем 15 словопозиций для ключа «мир».

Ассоциативный массив состоит в данном случае из двух компонент:

1) Файл данных. Хранит списки словопозиций. Словопозиции для одного ключа должны храниться преимущественно последовательно.

2) Словарь. Хранит в себе ключи и для каждого ключа информацию о том, где в файле данных располагаются словопозиции этого ключа.

In: A.A. Makhnev, S.F. Pravdin (eds.): Proceedings of the International Youth School-conference «SoProMat-2017», Yekaterinburg, Russia, 06-Feb-2017, published at http://ceur-ws.org

2 Способы создания инвертированных файлов

2.1 Обновление индекса

Обновлением индекса назовем ситуацию, когда на основании одного набора документов был создан индекс. Затем мы получили еще один набор документов, и требуется получить индекс, включающий объединенные данные первоначальных и новых документов.

2.2 Способ 1. Внешняя сортировка слиянием

Записываем в файл данных словопозиции, затем сортируем его таким образом, чтобы словопозиции, соответствующие одному ключу, шли подряд. При обновлении индекса, если появились новые документы, создается новый индекс и осуществляется процесс слияния предыдущего и нового индексов [1]. Чтобы не сливать новый и старый индекс каждый раз, можно поддерживать одновременно нескольких индексов и периодически сливать часть из них вместе, см., например, [2].

2.3 Способ 2. Легко обновляемые индексы

Словопозиции ключа хранятся в наборе блоков, которые могут располагаться в разных местах файла данных. Процесс создания индекса заключается в последовательном добавлении в индекс словопозиции, в процессе чего набор блоков для конкретного ключа может меняться, например, могут добавляться новые блоки. Обновление индекса осуществляется аналогично созданию индекса, т. е. заключается в последовательном добавлении в существующий индекс новых записей. В качестве примера можно рассмотреть [3, 4, 5].

При создании данного индекса требуется больше операций ввода-вывода. Это может быть решено различными способами организации кэш памяти и стратегиями формирования набора блоков. Также необходимо, чтобы блоки одного ключа располагались преимущественно подряд для снижения числа операций ввода-вывода при поиске. Алгоритмы, решающие данные проблемы, рассмотрены автором в [5, 6]. Преимущество данного способа в отсутствии процедуры слияния при обновлении индекса.

Первый и второй способы имеют достоинства и недостатки, для способа 1 при обновлении индекса требуется дорогостоящая операция слияния индексов, для способа 2 требуется существенно большее число дисковых операций при создании индекса.

Возможность более быстрого обновления индекса может быть важна. В [7, 8, 9, 10] автором рассматриваются несколько вариантов индексов, где ключом является не одно слово или его форма, а несколько форм слов. Эти дополнительные индексы могут применяться для значительного ускорения поиска, когда поиск осуществляется с учетом расстояния. В [8] рассмотрен алгоритм построения подобных индексов. Он включает в себя, в частности, разделение всего индексируемого массива текстов на части (размер которых зависит от объема доступной оперативной памяти, обычно порядка 10-20 ГБ) и последовательное добавление каждой части в индекс. В [7, 10] показано, что рассматриваемые дополнительные индексы позволяют повысить скорость выполнения некоторых видов поисковых запросов в десятки раз.

3 Структура легко обновляемого индекса

Таким образом, рассматриваемый нами индекс относится ко второму типу индексов. В данной работе рассматриваются стратегии создания легко обновляемых индексов и результаты экспериментов. Рассматривается новая стратегия создания цепочки блоков ограниченной длины, предназначенная для более быстрого обновления индекса.

Файл данных разбит на блоки фиксированного размера, называемые кластерами. Размер кластера задается при создании индекса, например, 32 КБ.

Потоком кластеров будем называть набор связанных кластеров, содержащий один список словопозиций.

Данные конкретного ключа хранятся в потоке кластеров. Примером потока кластеров является цепочка одиночных кластеров.

4 Цепочка одиночных кластеров

При добавлении первой словопозиции ключа в индекс мы можем выделять один кластер в нем. В кластере зарезервируем свободное место, необходимое для хранения ссылки (номера) на другой кластер.

Далее добавляем следующие словопозиции ключа в этот кластер последовательно. Когда свободное место в кластере заканчивается, выделяем еще один кластер, прописываем в текущем кластере ссылку на новый кластер и начинаем помещать данные в новый кластер.

На рис. 1 изображены три кластера, два из которых заполнены полностью, а последний – частично. В заполненных кластерах хранится ссылка на следующий кластер. Добавление данных всегда осуществляется в последний кластер цепочки. В словаре сохраняется номер первого и последнего кластера цепочки. Черный прямоугольник в конце кластера обозначает область, которая хранит ссылку на следующий кластер.

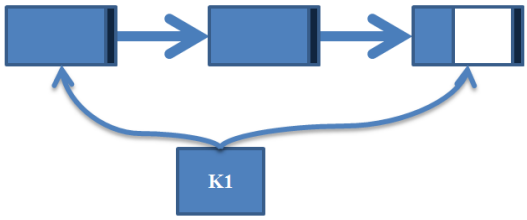

Рис. 1: цепочка одиночных кластеров. Ключ K1 ссылается на первый и последний кластер.

Однако, просто создавать цепочки кластеров нельзя, так как число операций чтения при поиске должно быть ограничено. Если каждый кластер будет в другом месте диска, то будет слишком много операций ввода-вывода. Поэтому данный способ не используется. Рассмотрим несколько более сложных стратегий, позволяющие создавать такой вид индекса.

5 Стратегии создания легко обновляемого индекса

5.1 Стратегия C1

Для каждого потока кластеров храним не менее одного кластера в памяти во время создания индекса. Поскольку ключей много, к примеру, в словаре русского языка около 200 тыс. базовых форм, их можно разделить на группы, например, по 2000 ключей, и данные каждой группы добавлять в индекс последовательно (как описано в [5, 8]). Один кластер в памяти для каждого потока кластеров является необходимым условием. В целях повышения производительности объем этого кэша может быть больше, например, из расчета по 10-15 кластеров для каждого потока.

Обозначение C1 образовано от слова *cluster* и 1.

5.2 Стратегия EM

При малом размере данных словопозиции ключа могут храниться прямо в словаре, вместе с ключом. Обозначение EM образовано от английского слова *embedded*.

5.3 Стратегия PART

Если объем данных ключа меньше, чем половина кластера, то можно использовать часть кластера. Кластер делится на равные части, каждая из которых может использоваться каким-то ключом. Вводим виды кластеров, разделенные на 2 части, на 4, на 8 и т.д. Вначале используем кластер с минимальным размером части. Когда свободное место заканчивается, переносим данные в кластер с большим размером части. Когда размер данных превысит половину кластера, переносим данные в целый кластер. Детально см. в [5].

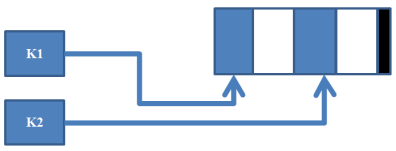

Рис. 2: кластер, разделенный на части

На рис. 2 изображено, как два ключа используют каждый по одной части кластера, разделенного на 4 части. Две остальные части свободны. Черным цветом показана располагающаяся в конце кластера область метаданных, в которой указано, какие части кластера заняты. Кластеры, разделенные на части, называем PART-кластерами. Обозначение PART образовано от английского слова *part*.

5.4 Стратегия S

Если данных больше, чем 1 кластер, то вводим понятие сегмента кластеров. Сегмент – это несколько последовательно располагающихся кластеров. Используем сегменты, число кластеров в которых – степень 2-х. Выделяем для данных сегмент и пишем в него последовательно. Если свободное место заканчивается, выделяем сегмент вдвое большего размера, переносим в него данные, в первую половину. Параметром N определяем максимальный размер сегмента. Если сегмент максимального размера заполнен, создаем новый сегмент. При этом, если в наборе кластеров уже существует сегмент максимального размера, все последующие сегменты также создаем максимального размера. Детально см. в [5].

Пусть $N = 8$. На рис. 3 изображен поток кластеров, состоящий из трех сегментов. Два первых сегмента полностью заполнены. В последнем заполнены полностью два первых кластера, а третий – частично. Последние кластеры первых двух сегментов содержат ссылку на первый кластер следующего сегмента. На практике размер максимального сегмента выбирается не менее нескольких десятков МБ. (несколько сотен кластеров и более). Обозначение S образовано от английского слова *segment*.

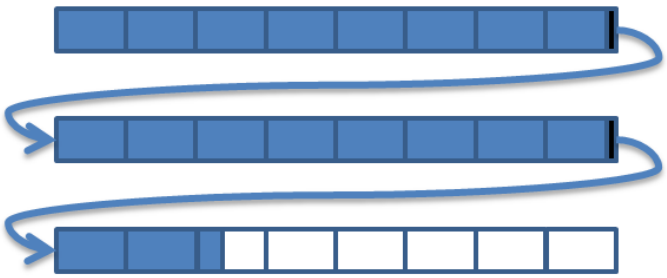

Рис. 3: сегменты кластеров

5.5 Стратегия FL

Сформируем отдельную область последовательно располагающихся кластеров. Для конкретного ключа выделяем в ней один кластер, назовем его FL-кластером. Новые записи помещаем последовательно в этот кластер.

Данная стратегия применяется совместно с другими, например, с S. Когда свободное место в FL-кластере заканчивается, данные из него переносятся в сегмент, а FL-кластер очищается и используется далее. Преимущество: при обновлении индекса область FL-кластеров может быть быстро считана в кэш. Таким образом, последние кластеры для каждого ключа будут загружены в кэш. На рис. 4 изображен поток кластеров, состоящий из двух сегментов. Один сегмент полностью заполнен. Во втором сегменте заполнено пять кластеров. Данные добавляются в дополнительный FL-кластер, который частично заполнен. Для ключа K1 в словаре сохранены ссылки на первый кластер потока, последний заполненный кластер последнего сегмента и FL-кластер.

Также, для стратегии PART имеет смысл выделять PART-кластер в области FL-кластеров. Детально см. в [11].

Обозначение FL образовано от английских слов *first* и *level*. Можно сказать, что мы делим кластеры на два подмножества (уровня): те, которые скорее всего будут обновлены (первый уровень), и остальные, данные в которых более статичные (второй уровень).

5.6 Стратегия TAG

Данные нескольких ключей могут одновременно храниться в одном потоке кластеров. Для этого к каждой словопозиции добавляется тег – номер ключа.

Допустим, есть два ключа:

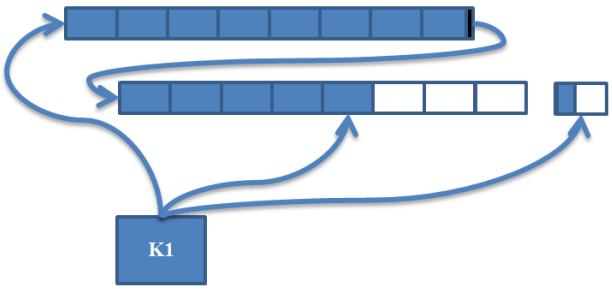

Рис. 4: сегменты кластеров с дополнительным FL-кластером

K1, со списком словопозиций: (D1, P1), (D2, P2), ... ,
K2, со списком словопозиций: (F1, Q1), (F2, Q2), . . .
Можно сформировать список словопозиций:

$$(D1, P1, 1), (F1, Q1, 2), (F2, Q2, 2), (D2, P2, 1), \ldots,$$

где в каждую исходную словопозицию добавлен номер ключа, 1 для K1, 2 для K2. Нумерация ключей определена локально в пределах данного списка. Данная последовательность хранится в одном потоке кластеров. Словопозиции в ней отсортированы по тем же критериям, как в исходных списках. Таким образом, два (или более) ключа ссылаются на один поток кластеров. Соответствующий ключу номер (тег) хранится в словаре.

Длина подобного потока кластеров ограничена. Если для какого-то ключа объем данных превышает заданный порог, то его данные извлекаются и сохраняются в новом, выделенном специально для этого ключа потоке, в который уже словопозиции других ключей не помещаются и теги отсутствуют. Детально см. в [11, 12].

5.7 Стратегия CH

5.7.1 Цепочка кластеров с обратными ссылками ограниченной длины

В каком-то смысле вернемся к простому варианту. Используем цепочку одиночных кластеров. Но ограничиваем ее размер. Добавляем данные в последний кластер. Если свободное место в последнем кластере заканчивается, добавляем в цепочку новый кластер, прописывая сразу в нем ссылку на предыдущий. Если кластеров становится много (например, больше 9), то преобразуем цепочку кластеров в сегмент кластеров и переходим к стратегии S. То есть выделяем пустой сегмент, считываем все кластеры цепочки, сохраняем данные в новом сегменте, а кластеры цепочки помещаем в список свободных кластеров для переиспользования. Стратегия CH не рассмотрена ранее.

Заметим, что есть два варианта организации цепочки кластеров. В начале статьи была приведена цепочка кластеров, когда мы в кластере указываем ссылку на следующий кластер цепочки. В этом же разделе (рис. 5) мы в кластере указываем ссылку на предыдущий кластер цепочки. Это позволяет уменьшить число дисковых операций при записи данных, но чтение данных требуется осуществлять с конца цепочки кластеров. Это означает, что при поиске вся цепочка кластеров должна быть считана в память для доступа к первой словопозиции. Это всегда возможно, так как длина цепочки ограничена.

Рис. 5: цепочка кластеров с обратными ссылками

Обозначение CH образовано от английского слова *chain*.

5.7.2 Цепочка сегментов с обратными ссылками ограниченной длины

Если при добавлении нового кластера в цепочку какое-то количество предыдущих кластеров оказывается в кэше, мы можем содержимое этих кластеров перенести в новое место, располагая друг за другом, см.

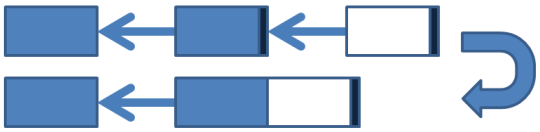

Рис. 6: слияние блоков в цепочке блоков

рис. 6. Таким образом, мы можем получить не цепочку кластеров, а цепочку сегментов кластеров. Момент перехода к стратегии S зависит не от числа кластеров в цепочке, а от числа сегментов в ней, потому что число сегментов определяет число обращений к разным местам диска при чтении данных этой цепочки.

Далее опишем, как в общем случае выглядит алгоритм добавления нового кластера в цепочку.

1) Идем назад от конца текущей цепочки и формируем список сегментов цепочки, каждый из которых полностью находится в кэше. Если какой-то кластер сегмента не находится в кэше, останавливаемся.

2) Копируем содержимое сегментов из кэша в отдельный буфер.

3) Выделяем новый сегмент большого размера (размер такой, чтобы поместить в него данные сегментов цепочки, скопированные на шаге 2 и данные нового кластера).

4) Скопированные на шаге 2 сегменты освобождаем.

5) В новом большом сегменте прописываем ссылку на последний сегмент цепочки, который мы не сливаем.

6) Записываем в новый сегмент данные сливаемых сегментов и данные нового кластера. Было выявлено, что имеет смысл сливать не менее двух сегментов (плюс новый кластер). Если сливать просто один последний сегмент и новый кластер, то образуется слишком много свободных сегментов, которые накапливаются (на шаге 4 мы освобождаем старые сегменты, что означает, что их адреса помещаются в список свободных сегментов, и они могут быть повторно использованы).

5.7.3 Ограничение на длину цепочки сегментов

Это ограничение должно определяться, исходя из ограничения на суммарное число операций ввода-вывода, выполняемых при поиске. Эксперименты показали, что при выборе числа 9 время поиска не изменяется, по сравнению с проведенными в [10] экспериментами. Имеет смысл также задать для разных потоков кластеров разные граничные условия. Например, выбирать для конкретного потока кластеров ограничение длины из диапазона 7-9 случайным образом или в зависимости от адреса первого кластера. Этот подход позволяет увеличить вероятность того, что свертывание цепочки сегментов (переход от CH к S) будет происходить для разных потоков кластеров при выполнении разных обновлений индекса.

5.8 Стратегия SR

Модификация стратегии FL. Вместо выделения специальной области в индексе, используем отдельный индекс – индекс коротких записей (ИКЗ). Потоку кластеров может соответствовать запись в ИЗК, назовем ее SR-записью. Эта запись представляет собой список словопозиций и хранится в наборе малых блоков (например, 128 байта каждый).

Размер данных SR-записи ограничен размером кластера. Как только данных становится больше, переносим данные из SR-записи в ее поток кластеров.

При завершении обработки группы ключей, данные ИКЗ сохраняются в специальном файле. При старте обновления индекса все данные ИКЗ считываются в память. Чтение и запись данных ИКЗ осуществляется последовательно с буферизацией.

Объем кэша ИКЗ ограничен. Если потоков кластеров становится слишком много, для новых потоков стратегия перестает применяться. Эта стратегия не рассмотрена ранее.

Цель данной стратегии – оптимизировать хранение данных в FL-кластерах. Если для потока кластеров мы выделяем целый FL-кластер, то мы не знаем, насколько он будет заполнен. Может быть, FL-кластер будет заполнен наполовину, или даже меньше, но записать этот кластер в файл мы должны целиком при

завершении обновления индекса. То есть объем ввода-вывода получается больше, чем объем хранимых данных. В ИКЗ же применяем малые блоки, размер которых существенно меньше размера кластера.

Применение стратегии SR вместе с CH позволяет избежать чтения кластеров цепочки при обновлении индекса. Мы стремимся держать кластеры цепочки полностью заполненными. Как только мы накопили в SR-записи объем, превышающий размер кластера, мы переносим в цепочку кластеров ровно столько данных, чтобы последний кластер цепочки был полностью заполнен (то есть, всегда добавляем в цепочку новый, полностью заполненный кластер). Если при этом в SR-записи было больше данных, чем требовалось для заполнения свободного места в последнем кластере цепочки, остаток оставляем в SR записи. Поэтому при каждом переносе данных в цепочку кластеров нам не требуется читать последний кластер цепочки, так как он: 1) уже заполнен, 2) мы сохраняем ссылку на предыдущий кластер в новом, а не наоборот.

На рис. 7 изображена цепочка сегментов с обратными ссылками и SR-записью, состоящей из четырех малых блоков (эти блоки при добавлении данных находятся в оперативной памяти). Ключ K1 ссылается на первый кластер цепочки, последний заполненный кластер и на SR-запись.

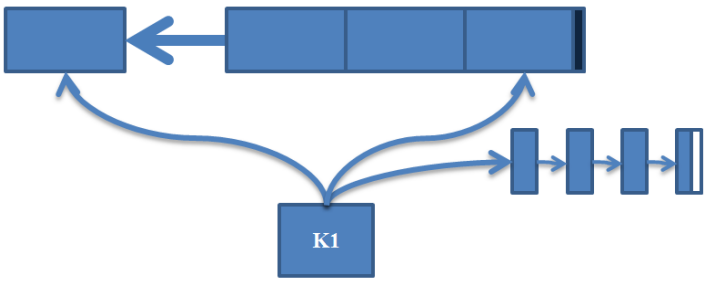

Рис. 7: цепочка кластеров с SR-записью

Обозначение SR образовано от английских слов *short*, *record*.

5.9 Стратегия DS

Используем два файла, один для больших дисковых операций, другой для малых. При этом при записи малых дисковых операций упаковываем их в большой буфер и сохраняем его (требуется также хранить таблицу исходных и физических адресов). Несколько дисковых операций, с адресами A, B, C, упаковываются в один большой буфер и получают в нем адреса a, b, c. Отдельно есть таблица преобразования: A $\rightarrow$ a, B $\rightarrow$ b, C $\rightarrow$ c. Детально см. в [9].

Обозначение DS образовано от *distributed store*.

5.10 Диаграмма состояний потока кластеров

На рис. 8 показано, как текущая стратегия хранения данных в потоке кластеров может меняться в зависимости от настроек и объема данных.

Стратегия TAG не включена в рисунок, так как она реализуется на уровне словаря.

Стратегия DS реализуется на уровне организации записи и чтения в файл индекса, и поэтому также не отображена на рисунке.

Стратегия C1 применяется неявно для каждой из стратегий, изображенных на рисунке.

В зависимости от объема данных мы можем начать использовать стратегию EM. Как только данных становится больше, переходим к стратегии PART либо используем SR (при этом на каком-то этапе может использоваться только SR, а кластер в файле с данными не выделяться).

В случае выбора стратегии PART, далее, когда объем данных превысит размер части для кластера, разделенного пополам, мы используем универсальную стратегию S либо промежуточную стратегию CH. Стратегия FL может применяться в качестве оптимизации. Обозначение S+FL означает применение одновременно стратегии S как основной и FL – как дополнительной.

В случае выбора SR мы далее выбираем либо S, либо CH, с применением SR в качестве оптимизации. PART в этом случае не используется, так как если мы уже решили хранить для некоторого потока кластеров отдельно в индексе коротких записей объем данных размером до размера одного кластера, то использовать PART смысла нет, так как она применяется для объемов, не превышающих один кластер.

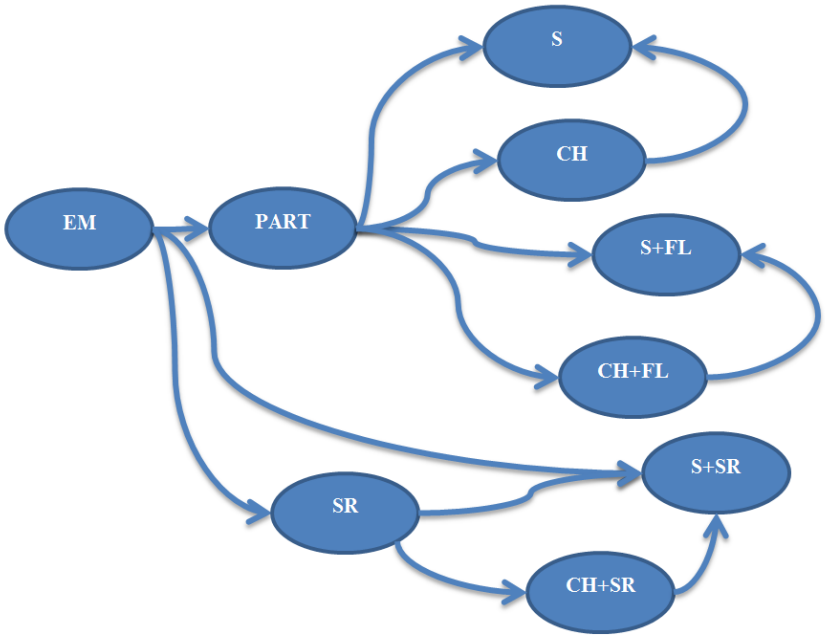

Рис. 8: диаграмма состояний потока кластеров

6 Применение легко обновляемых индексов для полнотекстового поиска

6.1 Виды запросов

Рассмотрим несколько запросов.
Time and a world Yes.
The Who who are you.
End of days.
Скажи мне кто твой друг.

Слова в текстах встречаются с разной частотой. Наибольшую сложность представляют запросы, включающие часто встречающиеся слова. Запрос с такими словами, как “and”, “who”, может выполняться долго с использованием обычного индекса, если ключ – это форма слова, а значение – список вхождений данного слова в текстах.

Если использовать дополнительные индексы, где ключом является несколько форм слов, выполнение ряда запросов можно ускорить в десятки раз [7, 10].

6.2 Три вида слов

Разделим все слова на 3 группы.

1) стоп слова, например: предлоги, местоимения,

2) часто используемые слова,

3) остальные.

Используем морфологический анализатор. Для каждой словоформы, входящей в словарь анализатора, он возвращает список номеров базовых форм слов. Такое слово мы назовем известным словом. Номер базовой формы – это число в диапазоне от 0 до WordsCount – 1, где WordsCount – число различных базовых форм (около 260 тыс. для используемого словаря).

Если слово не входит в словарь анализатора, то будем считать, что его базовая форма совпадает с самим словом. Такое слово назовем неизвестным.

Базовые формы слова также называются леммами, а сам процесс получения набора лемм по словоформе – лемматизацией. Разделение слов на три группы также применяется и к леммам.

6.3 Три вида индексов

В проведенных экспериментах использовалась структура индексов, описанная в [10]. Применяем три вида индексов.

1) Обычный индекс. Ключ – лемма слова.
2) Расширенный индекс. Ключ – пара лемм (w,v) (словопозиция формируется, когда слова располагаются близко в тексте).
3) Индекс последовательностей стоп лемм. Ключ – последовательность стоп лемм. Ключ формируется, когда в тексте встречается последовательность подряд идущих стоп лемм.

6.4 Описание экспериментов

Проиндексировано 71.5 ГБ текста. Далее приведены результаты трех экспериментов создания индексов. В каждом из трех экспериментов применялся разный набор стратегий:

1) C1+EM+PART+S+FL+TAG (первые 6 стратегий).
2) Все, что в 1, и CH+SR.
3) Все, что в 2, и DS.

Суммарный размер индекса в каждом случае – около 400 ГБ. В каждом эксперименте измеряются следующие показатели:

1) Суммарный объем прочитанных и записанных данных.
2) Суммарное число операций ввода-вывода.

Объем документов был разделен на две части. Вначале создан индекс для первой части. Затем в него добавлена вторая часть (обновление индекса).

Таблица 1: параметры

Параметр	Значение
Размер кластера	32 КБ
Число кластеров в кэше на один поток кластеров	45
Критерий малой операции ввода вывода (для DS, [9])	$\leqslant$ 32 КБ
Размер кэша файла данных (кластеров)	1 ГБ
Максимальная длина в сегментах цепочки кластеров для CH	9
Число групп известных лемм (вычисляемый параметр в зависимости от объема кэша и числа кластеров в кэше на один поток кластеров)	243
Число групп неизвестных лемм	96

Остальные параметры были как в [10].

6.5 Результаты экспериментов

Данные в таблице 2 показывают, что при применении дополнительных стратегий объем ввода-вывода существенно снижается. При этом стратегия DS влияет на этот показатель несущественно.

Данные в таблице 3 показывают, что при применении дополнительных стратегий количество операций ввода-вывода существенно снижается. При этом применение стратегии DS дополнительно к CH+SR позволяет добиться еще лучших результатов.

7 Заключение и выводы

В дополнение к ранее рассмотренным стратегиям организации легко обновляемого индекса рассмотрены дополнительные стратегии, позволившие ускорить создание индекса за счет снижения числа операций ввода-вывода. Результаты экспериментов в работе даны в привязке к рассмотренной ранее задаче полнотекстового поиска с учетом расстояния (proximity search), решаемой с помощью применения нескольких индексов с разными видами ключей.

Таблица 2: объем ввода-вывода в ГБ

Вид индекса	Номер эксперимента		
	1	2	3
Обычный индекс известных лемм	209,53	190,126	192,516
Обычный индекс неизвестных лемм	29,571	24,16	23,774
Расширенные индексы, w, v – известные	249,555	109,731	111,04
Расширенные индексы, w – известная, v – неизвестная	50,511	26,962	28,279
Индекс последовательностей стоп-лемм	359,023	310,645	320,552

Таблица 3: количество операций ввода-вывода

Вид индекса	Номер эксперимента		
	1	2	3
Обычный индекс известных лемм	139 694	86 115	62 993
Обычный индекс неизвестных лемм	23 670	12 919	9 085
Расширенные индексы, w, v – известные	126 251	38 228	11 662
Расширенные индексы, w – известная, v – неизвестная	50 098	33 618	26 010
Индекс последовательностей стоп-лемм	190 995	87 817	79 399

Список литературы

[1] J. Zobel, A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), Article 6, 2006.

[2] N. Lester, A. Moffat, J. Zobel. Fast On-Line Index Construction by Geometric Partitioning. *In CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, 776-783, 2005.

[3] A. Tomasic, H. Garcia-Molina, K. Shoens. Incremental updates of inverted lists for text document retrieval, *In Proc. ACMSIGMOD Int. Conf. on the Management of Data*, Minneapolis, Minnesota, 289-300, 1994.

[4] E. W. Brown, J. P. Callan, W. B. Croft. Fast incremental indexing for full-text information retrieval. *In Proceedings of the International Conference on Very Large Databases*, 192-202, Santiago, Chile, 1994.

[5] A. B. Veretennikov. Effective Indexing of Text Documents using CLB-Trees. *Control systems and information technologies*, 35(1.1): 134-139, 2009 (in Russian). = А. Б. Веретенников. Эффективная индексация текстовых документов с использованием CLB-деревьев. *Системы управления и информационные технологии*, 35(1.1): 134-139, 2009.

[6] A. B. Veretennikov. About phrases search in full-text index. *Control systems and information technologies*, 48(2.1): 125-130, 2012 (in Russian). = А. Б. Веретенников. О поиске фраз и наборов слов в полнотекстовом индексе. *Системы управления и информационные технологии*, 48(2.1): 125-130, 2012.

[7] A. B. Veretennikov. Using additional indexes for fast full-text searching phrases that contains frequently used words. *Control systems and information technologies*, 52(2): 61-66, 2013 (in Russian). = А. Б. Веретенников. Использование дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова. *Системы управления и информационные технологии*, 52(2): 61-66, 2013.

[8] A. B. Veretennikov. Creating additional indexes for fast full-text searching phrases that contains frequently used words. *Control systems and information technologies*, 63(1): 27–33, 2016 (in Russian). = А. Б. Веретенников. Создание дополнительных индексов для более быстрого полнотекстового поиска фраз, включающих часто встречающиеся слова. *Системы управления и информационные технологии*, 63(1): 27–33, 2016.

[9] A. B. Veretennikov. Using additional indexes of frequently used words for full-text search. *Data Analytics and Management in Data Intensive Domains XVIII International Conference, DAMDID/RCDL 2016*, 217-224.

Ershovo, Moscow, Russia, October 11-14, 2016 (in Russian). = А. Б. Веретенников. О применении дополнительных индексов часто встречающихся слов для полнотекстового поиска. *Аналитика и управление данными в областях с интенсивным использованием данных. XVIII Международная конференция DAMDID / RCDL'2016*, 217-224, 11-14 октября 2016 года, Ершово, Москва.

[10] A. B. Veretennikov. Efficient full-text search by means of additional indexes of frequently used words. *Control systems and information technologies*, 66(4): 52-60, 2016 (in Russian). = А. Б. Веретенников. Эффективный полнотекстовый поиск с использованием дополнительных индексов часто встречающихся слов. *Системы управления и информационные технологии*, 66(4): 52-60, 2016.

[11] A. B. Veretennikov. On CLB-tree building method optimization. *Proceedings of the 41th all-russian Youth School-conference "Modern Problems in Mathematics and its Applications"* 429-435, Yekaterinburg, 2010 (in Russian). = А. Б. Веретенников. О методе оптимизации создания CLB-дерева. *Проблемы теоретической и прикладной математики: Тезисы 41-й Всероссийской молодежной конференции*, 429-435, Екатеринбург: УрО РАН, 2010.

[12] A. B. Veretennikov. On building easy updatable full-text indexes. *Digital libraries: advanced methods and technologies, digital collections. The Tenth Anniversary of All-Russian Research Conference* 149-154, Dubna, 2008, (in Russian). = А. Б. Веретенников. Создание легко обновляемых текстовых индексов. *Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды Десятой Всероссийской научной конференции "RCDL'2008",* 149-154, Дубна: ОИЯИ, 2008.

About a structure of easily updatable full-text indexes

Alexander B. Veretennikov
Ural Federal University (Yekaterinburg, Russia)

Keywords: full-text search, search engines, inverted files, additional indexes.

We consider strategies of organization of easily updatable associative arrays in external memory. These arrays are used for full-text search. We study indexes with different keys: single word form, two word forms, sequence of word forms. Structure of storage depends on the size of key's data. Results of the experiments are given in the context of the proximity full-text search by means of additional indexes.