Полнотекстовый индекс для часто обновляющихся библиотек

© Веретенников А.Б.

Уральский государственный университет им. А.М.Горького
alexander@veretennikov.org

Аннотация

Автор ведет разработку алгоритмов создания полнотекстового индекса текстовых документов. Как правило, подобные индексы строятся в виде инвертированных файлов. При этом полученный индекс трудно обновлять, т. к. для добавления новых данных в индекс его требуется практически переписать его заново. Структура данных, разработанная автором [1-12] позволяет создавать индекс, который может легко и быстро обновляться. Она весьма полезна при создании индексов быстро обновляющихся текстовых документов. В данной работе описан метод, позволяющий улучшить скорость обновления индекса.

В статье рассматривается вопрос построения и быстрого обновления индекса. При этом обосновывается то, что скорость поиска не будет снижена.

1 Введение

В [4,6,11] автором была описана структура данных CLB-индекс и алгоритмы ее построения. Даны теоретические оценки, подтверждающие эффективность CLB-индекса и результаты экспериментов. В [9] описан подход, позволивший существенно улучшить скорость обновления индекса.

Основное достоинство разработанной структуры данных является возможность быстрого добавления в индекс новых данных. Это актуально во многих областях, например при создании текстовой базы новостей или в корпоративных системах, где требуется быстрая доступность информации. Чем раньше важный документ станет доступен пользователям в поиске, тем лучше. В идеале документ должен быть доступен в поиске сразу после своего создания.

Труды 13[ой] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2011, Воронеж, Россия, 2011.

Инвертированные файлы [22, 25] или их аналоги [18, 21], построенные путем применения сортировки данных и записи отсортированных данных в файл, сложны в обновлении. Для добавления данных в индекс требуется переписать весь индекс. Как правило, в документации систем полнотекстового поиска часто используются термины вида «Слияние индекса» или «Merge», т. к. для решения проблемы обновления индекса создаются несколько индексов и их слияния делаются как можно реже, т. к. являются затратными операциями.

Можно сказать, что инвертированный файл это кирпич, и для добавления в него чего-то его требуется разделить обратно до песка и затем, смешав с новым песком, собрать снова. В отличие от этого, в CLB индексе за счет алгоритма создания индекса остаются пустые места, для помещения в них новых данных, можно сравнить его с губкой, которая может легко впитать в себя новые данные.

Следует сказать, что вопросы быстрого обновления инвертированных файлов ранее рассматривались различными исследователями. Инвертированный файл представляет собой набор записей вида (*ID*, *P*), где *ID* – идентификатор документа, *P* – позиция, например порядковый номер, слова в документе. Все записи, соответствующие одному слову хранятся последовательно для их быстрого чтения при поиске. Например, пусть нужно объединить два индекса, один из которых основной и имеет большой размер, а второй представляет собой индекс новых данных, имеет меньший размер. Рассмотрим далее несколько вариантов.

В [19] предлагается для каждого слова из второго индекса выполнить следующее: прочитать все записи для этого слова в первом индексе, объединить записи с записями второго индекса и записать полученные данные в первый индекс. Соответственно, рассматриваются вопросы управления свободным местом, которое появляется в первом индексе в результате перемещения записей в новое место, т. е. его повторным использованием. В качестве оптимизации рядом авторов предлагается при создании индекса после записей определенного слова резервировать свободное место, чтобы была возможность в определенных

случаях добавлять туда новые данные. Например, в [23] для определения объема резервируемого свободного места применяется вероятностей подход.

Следует отметить, недостатки данного подхода, а именно:

1) Для каждого слова обрабатываются все его записи, и их объем никак не ограничен, т. е. в худшем случае придется обработать весь индекс.
2) Большое количество операций случайного доступа к диску, которое достигает числа различных слов в новых документах.

В [15, 16] авторы предлагают другую структуру данных для инвертированного файла, где для хранения данных оперируют блоками размера до 8 кб. Данный подход обладает следующими недостатками:

1) Никак не контролируется возможная фрагментация индекса, что приводит к падению скорости поиска.
2) Опять же, большое число операций случайного доступа к диску.

Такие же проблемы присутствует и в [24].

Во всех выше описанных случаях при обновлении индекса возникает много операций случайного доступа к диску. Это существенно замедляет создание индекса. В частности в [19, 20] делается вывод, что слияние индексов (merge), как правило, существенно быстрее частичного обновления индекса (in-place update). Это вызвано тем, что при слиянии индексов доступ к диску осуществляется последовательно, а при частичном обновлении превалирует случайный доступ к диску.

В [17] рассматривается смешанный подход, когда для редко встречающихся слов применяются слияния (merge) индексов, а для часто встречающихся слов применяются методы частичного обновления индекса.

CLB-индекс не обладает указанными недостатками. Некоторые свойства CLB индекса:

1) За счет особых методов кеширования и организации структур данных существенно снижается число дисковых операций.
2) Контролируется последовательное расположение записей для одного слова на диске, в результате при поиске чтение в основном последовательное.

В результате CLB-индекс как создается, так и обновляется быстро, существенно быстрее инвертированных файлов. По скорости поиска CLB-индекс не уступает инвертированным файлам.

Однако, из приведенных в [9] результатов экспериментов ясно просматривается определенное торможение на малых объемах данных, до 50 – 100 Мб текста. Подобная проблема имеет практически очевидное решение, описанное далее в данной работе.

2 Обновление индекса раньше

Для иллюстрации проблемы автор приводит результаты эксперимента по созданию и обновлению индекса.

Эксперимент заключается в создании первоначального индекса на основании некоторого массива документов, и далее добавление в индекс данных различного размера:

1) Создание индекса для базового массива документов.
2) Добавление в индекс небольшого файла.
3) Добавление в индекс 1, 2, 3, ..., 9 Мб данных.
4) Добавление в индекс 10, 20, 30, ..., 90 Мб данных.
5) Добавление в индекс 100, 200, 300, ..., 900 Мб данных.
6) Добавление в индекс 1000, 2000, 3000, ..., 10000 Мб данных.

В представленных результатах видно, что CLB индекс более эффективен при добавлении в него значительного количества данных (от 1 Гб) чем небольшого (1 – 10 Мб), что связано с некоторыми постоянными накладными расходами при каждой операции добавления данных в индекс.

Как показано в приведенной далее таблице, добавление данных небольшого размера занимает 1 – 5 минут. При добавлении данных в инвертированный файл это могло быть занять часы при достаточно большом объеме индекса. Тем не менее, хотелось бы ускорить обновление индекса при малых объемах данных.

Время обновления индекса обусловлено объемом изменяемых данных (не менее 3-х Гб), как описано в [9] этот объем не зависит от объема проиндексированных текстов, а обусловлен объемом словаря морфологического анализатора. При обновлении индекса объем прочитанных и записанных данных складывается из константной части (в таблице примерно 3 Гб), и части, размер которой линейно зависит от объема исходных данных. В конце таблицы видно, что при больших объемах данных суммарное количество прочитанных и записанных байт примерно в два раза больше объема исходных данных.

Далее в таблице, размер – размер текста, добавляемого в индекс, общее время – суммарное время, составленное из времени чтения файлов и времени записи индекса. В последних двух колонках указан суммарный объем прочитанных и записанных кластеров (блок данных в индексе, см. [1-12]). В таблице: К – Кб., М – Мб., Г – Гб.

Все обрабатываемые файлы представляли собой обычный текст, преимущественно художественная литература, журналы, статьи.

Информация об эксперименте:

1) Суммарный объем текстов: 143 Гб.
2) Всего документов 1 602 952.
3) Всего слов: 21 075 541 239
4) Слов, не входящих в словарь морфологического анализатора: 1 067 190 218 (6%).

5) Различных слов, не входящих в словарь морфологического анализатора: 32 161 992.
6) Суммарное время создания и обновления индекса: 9 ч. 46 мин.
7) Размер основного кеша 1 Гб (т. е. ограничение на объем данных, которые в процессе создания индекса хранятся в оперативной памяти).
8) Максимальное количество оперативной памяти, которое было задействовано: 924 Мб. Результат вызова функции *GetProcessMemoryInfo*: *PeakPagefileUsage* = 924 Мб, *PeakWorkingSetSize* = 911 Мб.

Размер	Общее время, ч.:м.:с.	Время записи индекса	Чтение кластеров	Запись кластеров
85.2 Г	03:16:06	01:18:04	7.9 Г	59.4 Г
1.4 К	00:48	00:43	3.3 Г	4.5 М
1.1 М	01:48	01:43	3.3 Г	321.5 М
2.2 М	02:09	02:04	3.3 Г	464.2 М
3.1 М	02:25	02:19	3.3 Г	582.2 М
4.0 М	02:32	02:26	3.3 Г	679.7 М
5.0 М	02:42	02:37	3.3 Г	746.1 М
6.3 М	02:45	02:39	3.3 Г	820.0 М
7.3 М	03:00	02:54	3.3 Г	864.1 М
8.2 М	02:56	02:50	3.3 Г	899.7 М
9.2 М	03:07	03:01	3.3 Г	956.6 М
10.0 М	03:10	03:04	3.3 Г	1003.6 М
10.0 М	03:10	03:05	3.3 Г	999.3 М
20.0 М	03:33	03:26	3.3 Г	1.3 Г
30.4 М	03:44	03:36	3.3 Г	1.4 Г
40.2 М	04:03	03:55	3.3 Г	1.6 Г
50.6 М	04:06	03:57	3.3 Г	1.7 Г
60.2 М	04:03	03:54	3.3 Г	1.7 Г
70.3 М	04:07	03:57	3.3 Г	1.7 Г
80.5 М	04:19	04:09	3.3 Г	1.8 Г
90.0 М	04:21	04:10	3.3 Г	1.8 Г
100.1 М	04:28	04:15	3.3 Г	1.8 Г
200.4 М	05:37	05:10	3.4 Г	2.3 Г
300.1 М	06:12	05:37	3.5 Г	2.5 Г
400.6 М	07:40	06:56	3.6 Г	2.6 Г
500.5 М	08:03	07:11	3.7 Г	2.8 Г
602.7 М	08:24	07:24	3.7 Г	2.9 Г
702.8 М	08:50	07:43	3.8 Г	3.1 Г
800.6 М	09:11	07:56	3.9 Г	3.2 Г
902.7 М	09:31	08:10	3.9 Г	3.3 Г
1.0 Г	10:01	08:31	4.0 Г	3.4 Г
2.0 Г	13:29	10:41	4.6 Г	4.6 Г
2.9 Г	18:03	13:47	5.1 Г	5.8 Г
3.9 Г	20:53	15:18	5.5 Г	6.8 Г
4.9 Г	23:44	16:47	5.9 Г	7.6 Г
5.9 Г	27:28	19:25	6.4 Г	8.8 Г
6.8 Г	30:53	21:13	6.8 Г	9.7 Г
7.8 Г	34:32	23:31	7.6 Г	10.9 Г
8.8 Г	37:18	25:08	7.8 Г	11.7 Г
9.8 Г	42:26	28:45	8.5 Г	12.9 Г

Табл. 1. Создание и обновление индекса.

Результаты получены на следующей конфигурации:

Процессор: Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz. Оперативная память: 12 Гб.

Жесткий диск: Seagate Barracuda 7200.11, 7200 RPM, кэш 32 Мб., объем 2 Гб, ST32000641AS.

ОС: Microsoft Windows 2008 Enterprise x64 Edition with Service Pack 2.

Файловая система: NTFS.

Другие результаты экспериментов создания и обновления индекса на различных конфигурациях (включая более слабые, например на основе Celeron D с 1 Гб RAM) представлены на сайте http://www.veretennikov.org.

3 Решение проблемы быстрого обновления индекса при малых объемах новых данных

3.1 Промежуточный индекс

Решение проблемы заключается в создании некоторого промежуточного индекса для небольших данных. Таким образом, мы имеем основной индекс, структура которого описана в [4,6,9,11], и промежуточный индекс, структура которого описана далее.

При поиске мы вначале будем искать в основном индексе, затем в промежуточном.

Требования к промежуточному индексу:

1) Быстрая запись данных в промежуточный индекс.
2) Время поиска в промежуточном индексе должно быть ограничено задаваемым параметром.
3) Суммарный размер данных, сохраняемых в промежуточном индексе, ограничен заданным параметром.

3.2 Структура основного индекса

Как описано в [1-12] при создании CLB-индекса используется морфологический анализатор. В качества примеров используемых анализаторов можно привести [13,14].

Для каждого слова, известного анализатору он возвращает одну или несколько базовых форм. Слова, входящие в состав данного анализатора составляют 80-90% от общего количества слов тестов. Количество форм слов в анализаторе – около 200 000, обозначим количество базовых форм *FormsCount*.

Под известными словами мы будем понимать слова, входящие в словарь морфологического анализатора, а под неизвестными — те слова, которые не входят в этот словарь. Известные слова могут иметь несколько базовых форм. Неизвестные слова всегда имеют одну базовую форму — само слово. Таким образом, каждому слову сопоставляется набор его базовых форм (далее

просто форм).

В основном индексе для каждого вхождения слова в каждом документе для каждой его базовой формы сохраняются записи с полями

1) *ID* документа.
2) Позиция в документе (например, номер слова по порядку или отступ в байтах от начала файла).

3.3 Структура промежуточного индекса:

Пусть *M* — некоторый параметр.

1) Если объем данных меньше чем *M*, то данные добавляются в промежуточный индекс.
2) Если объем данных больше или равен *M*, то вначале данные из промежуточного индекса переносятся в основной индекс, затем в основной индекс добавляются новые данные.

Объем промежуточного индекса будет фиксирован, и ограничен неким параметром *S*. При этом, имеет смысл вычислять *M* в зависимости от *S* и текущего объема данных в промежуточном индексе, например, *M* = (*S* – <*Объем данных в промежуточном индексе*>)/2. Предполагается, что *S* примерно равно 500 Мб ~ 1 Гб.

Промежуточный индекс должен обновляться существенно быстрее, чем основной, но при этом поиск в нем должен быть также быстрый. Это можно сделать, с учетом того, что промежуточный индекс имеет ограниченный объем.

Введем параметр *GroupDataSize*. Размер этого параметра, может быть, например 500 Кб – 2 Мб. Выберем также параметр *GroupSize*, например 500. Разделим все *FormsCount* базовых форм на группы по *GroupSize* форм в каждой. Соответственно, получим *GroupsCount* = *FormsCount* / *GroupSize* групп. Далее создадим файл размером *S* = *HeaderSize* + *GroupDataSize*× *GroupsCount* байт. Где *HeaderSize* некий заголовок, размером, например, 64 К. Будем считать, что данный файл разделен на несколько областей: вначале заголовок файла, затем *GroupsCount* областей размером *GroupDataSize* байт.

Информация о вхождениях слов из определенной группы будет помещаться в соответствующую область полученного индекса. Для каждого вхождения слова в файле мы помещаем в промежуточный индекс запись с полями:

1) *ID* документа.
2) Позиция в документе.
3) *ID* базовой формы.

Т. е. запись в промежуточном индексе является записью в основном индексе с добавленным ID базовой формы.

При этом, для данной записи определяем номер группы путем деления *ID* базовой формы на *GroupSize*, и запись добавляется в соответствующую данной группе область файла.

Если область определенной группы полностью заполняется, то все данные из этой области переносим в основной индекс, а область очищаем.

Таким образом, файл промежуточного индекса всегда имеет постоянный размер. Имеет смысл его создать заранее требуемого размера, заполнив нулями, чтобы избежать возможной фрагментации на уровне файловой системы.

3.4 Поиск в промежуточном индексе:

1) По *ID* базовой формы определяем номер ее группы (путем деления *ID* на *GroupSize).*
2) Читаем данные определенной на предыдущем шаге группы, учитывая только записи с требуемым *ID* базовой формы.

Заметим, что поиск информации для любой базовой формы в промежуточном индексе требует не более чем чтения *GroupDataSize* байт с диска.

3.5 Кеширование при записи данных в промежуточном индекса:

Естественно осуществлять запись в промежуточный индекс с использованием кеша. Для каждой группы создается буфер, и запись в область данных группы осуществляется только при заполнении этого буфера. Кеширование на уровне операционной системы отключается в той степени, в которой это возможно (в частности, при работе в Microsoft Windows при вызове функции *CreateFile* для открытия файла применяется флаг FILE_FLAG_NO_BUFFERING).

3.6 Результаты экспериментов.

Далее приведены результаты эксперимента, который был проведен на той же конфигурации что указана в разделе 2, с теми же параметрами и исходными данными, но при обработке данных о словах, входящих в словарь морфологического анализатора использовался промежуточный индекс.

Для слов, не входящих в словарь анализатора промежуточный индекс не использовался, что и обусловливает наличие ненулевых значений в колонках «Чтение кластеров», «Запись кластеров» для малых объемов данных (т. е. порядка 10 – 20 строк, начиная со 2-й). Применение промежуточного индекса для слов, не входящих в словарь анализатора требует определенной доработки, но не составляет трудности.

Информация об эксперименте:

1) Суммарное время создания и обновления индекса: 8 ч. 46 мин.
2) Установленный размер основного кеша 1 Гб (т. е. такое же, как без использования промежуточного индекса).
3) Максимальное количество оперативной памяти, которое было задействовано: такое же, как без использования промежуточного индекса.

Размер	Общее время, ч.:м.:с.	Время записи индекса	Чтение кластеров	Запись кластеров	Чтение (пром. индекс)	Запись (пром. индекс)
85.2 Г	03:14:29	01:16:44	7.9 Г	59.3 Г	18.2 М	49.8 М
1.4 К	00:14	00:08	456.0 М	72.0 К	96.0 К	81.5 К
1.1 М	00:23	00:18	456.0 М	4.8 М	145.5 К	847.5 К
2.2 М	00:32	00:26	458.3 М	10.4 М	408.5 К	1.6 М
3.1 М	00:37	00:32	456.0 М	11.1 М	412.0 К	2.1 М
4.0 М	00:38	00:32	456.0 М	13.6 М	420.5 К	2.7 М
5.0 М	00:40	00:34	457.1 М	16.3 М	426.5 К	3.2 М
6.3 М	00:42	00:36	458.4 М	21.1 М	427.5 К	3.9 М
7.3 М	00:45	00:39	456.6 М	21.9 М	452.0 К	4.5 М
8.2 М	00:47	00:41	460.4 М	27.6 М	456.5 К	5.1 М
9.2 М	00:48	00:43	457.7 М	26.6 М	458.0 К	5.6 М
10.0 М	00:53	00:47	461.6 М	30.3 М	459.5 К	6.0 М
10.0 М	00:48	00:42	457.2 М	27.1 М	460.5 К	6.0 М
20.0 М	01:05	00:59	458.4 М	41.5 М	459.5 К	11.5 М
30.4 М	01:15	01:07	463.2 М	51.6 М	483.0 К	17.3 М
40.2 М	01:30	01:22	472.9 М	74.9 М	2.4 М	22.9 М
50.6 М	01:39	01:30	474.7 М	78.2 М	2.4 М	28.8 М
60.2 М	01:47	01:37	490.3 М	111.2 М	7.2 М	34.4 М
70.3 М	01:55	01:45	508.7 М	129.7 М	12.9 М	40.1 М
80.5 М	02:26	02:15	581.3 М	231.9 М	25.3 М	45.9 М
90.0 М	02:42	02:30	617.9 М	258.6 М	34.9 М	51.4 М
100.1 М	03:03	02:50	666.7 М	300.5 М	43.5 М	57.0 М
200.4 М	05:04	04:37	938.3 М	624.3 М	78.3 М	107.0 М
300.1 М	06:48	06:13	1.4 Г	1.1 Г	147.0 М	145.7 М
400.6 М	08:03	07:18	1.8 Г	1.5 Г	175.6 М	152.6 М
500.5 М	08:17	07:23	2.1 Г	1.8 Г	187.5 М	160.4 М
602.7 М	08:47	07:46	2.2 Г	2.0 Г	186.4 М	188.9 М
702.8 М	08:53	07:45	2.6 Г	2.3 Г	176.2 М	148.1 М
805.1 М	09:18	08:02	2.7 Г	2.5 Г	166.8 М	170.6 М
902.7 М	09:23	08:01	3.0 Г	2.8 Г	146.6 М	111.3 М
1002.8 М	09:25	07:55	3.1 Г	2.8 Г	93.4 М	101.2 М
2.0 Г	12:57	10:05	4.2 Г	4.5 Г	105.0 М	40.9 М
2.9 Г	16:19	12:04	4.7 Г	5.6 Г	37.1 М	51.9 М
3.9 Г	19:15	13:43	5.2 Г	6.6 Г	46.5 М	64.4 М
4.9 Г	22:03	15:17	5.6 Г	7.5 Г	38.1 М	51.8 М
5.9 Г	25:48	17:45	6.1 Г	8.6 Г	47.1 М	45.3 М
6.8 Г	29:04	19:31	6.5 Г	9.5 Г	52.7 М	45.7 М
7.8 Г	32:12	21:21	7.3 Г	10.7 Г	41.3 М	46.3 М
8.8 Г	35:18	23:10	7.5 Г	11.5 Г	42.2 М	46.7 М
9.8 Г	40:07	26:36	8.2 Г	12.8 Г	58.4 М	48.4 М

Табл. 2. Создание и обновление индекса с использованием промежуточного индекса.

3.7 Сравнение результатов экспериментов.

Далее на графиках тонкая линия соответствует обновлению индекса без применения промежуточного индекса, более толстая линия, располагающаяся преимущественно ниже тонкой линии, соответствует обновлению индекса с применением промежуточного индекса.

Графики построены на основании таблиц, исключая первую строку, соответствующую первоначальному созданию индекса.

На нижней оси каждого графика каждое деление соответствует одной строке таблицы.

На первом графике отображено общее время для каждого обновления индекса. Примерно первые 20 порций данных с применением промежуточного индекса добавляются быстрее, а далее скорость обновления индекса примерно одинакова.

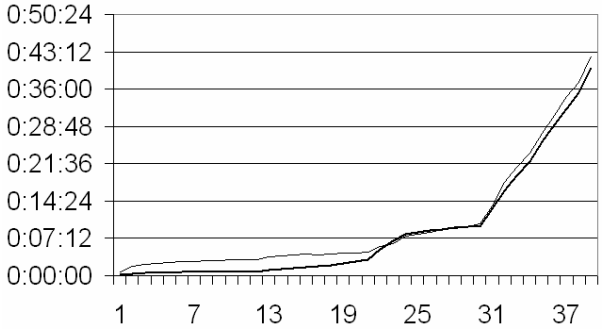

Рис 1. Время обновления индекса.

На втором графике показано общее время обновления индекса для первых двух десятков порций.

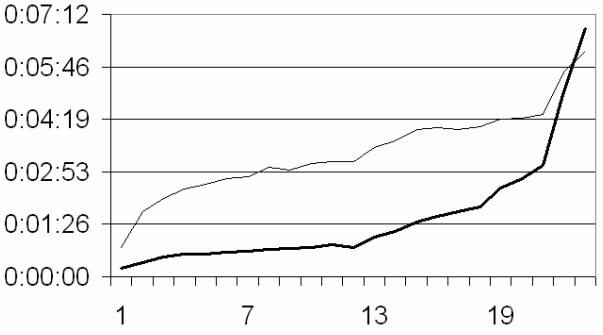

Рис 2. Время обновления индекса (малые объемы новых данных).

На третьем графике по вертикальной оси отображается суммарный объем записанных и прочитанных данных (в Гб) при добавлении конкретной порции данных в индекс.

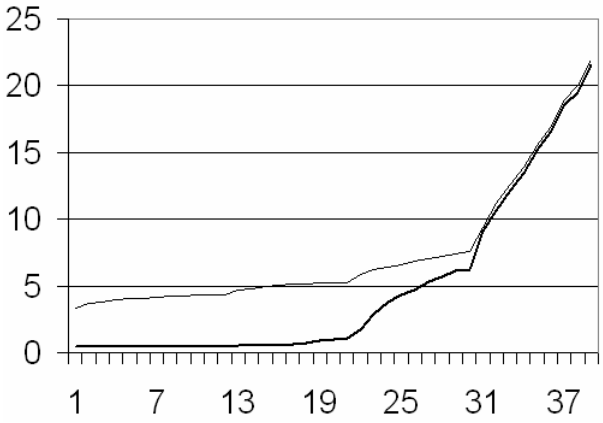

Рис 3. Обращение к диску при обновлении индекса.

На малых объемах данных мы видим из графиков и таблиц ускорение до 5 раз, а объем прочитанных/записанных данных меньше до 10 раз.

3.8 Комментарий про инвертированные файлы.

Данный подход и слияния инвертированных файлов принципиально различны, т. к. последние осуществляются при достижении всех сливаемых индексов большого объема, иначе слияние вырождается в перезаписывание большего из индексов. Описанный же «промежуточный индекс» предназначен для хранения данных малого объема. Это в каком-то смысле просто некоторый кеш для одного большого «основного индекса».

3.9 Дополнительные расходы при поиске

В приведенных экспериментах параметр *GroupDataSize* равен 1 Мб. Таким образом, дополнительно для каждой базовой формы при поиске требуется прочитать не более 1 Мб. (при скорости последовательного чтения примерно 70 Мб/сек. на используемых HDD). *GroupSize* равен 500, суммарный размер промежуточного индекса примерно 530 Мб.

При необходимости для получения большей скорости поиска *GroupDataSize* можно уменьшить, в том числе за счет уменьшения параметра *GroupSize*. Автором были проведены эксперименты с *GroupSize* = 150, *GroupDataSize* = 128 Кб, суммарный объем объем промежуточного индекса примерно 230 Мб. Существенного снижения скорости обновления индекса не обнаружено. Меньшее значение *GroupDataSize* обеспечивает меньшие дополнительные расходы при поиске.

Однако при уменьшении *GroupDataSize* приходится уменьшать и *GroupSize*, чтобы суммарный размер промежуточного индекса был достаточен. Суммарный размер промежуточного индекса влияет на то, при каких объемах индексируемых новых данных он будет использоваться. В рассматриваемом случае скорость обновления индекса существенно падает при объемах 50-100 Мб текста, промежуточный индекс 100-200 Мб для таких объемов достаточен.

Но, уменьшение *GroupSize* ведет к увеличению *GroupsCount*, что определяет количество дисковых операций при добавлении данных в промежуточный индекс. Преимущество промежуточного индекса в малом количестве дисковых операций при обновлении, порядка 1 тыс., по сравнению с минимум 50 – 200 тыс. для основного индекса (зависит от количества базовых форм слов, объема добавляемых данных и различных настроек).

Тем не менее, более важно уменьшить *GroupDataSize*, чтобы дополнительные расходы при поиске были минимальны, при достаточной скорости обновления индекса. За счет этого дополнительные расходы при поиске достаточно малы, чтобы их можно было вообще не учитывать.

4 Заключение

В статье описано решение задачи быстрого обновления полнотекстового индекса, как для больших, так и для малых объемов исходных данных. При этом скорость поиска не снижается.

Литература

[1] Веретенников А. Б., Лукач Ю. С. CLB-деревья: новый способ индексации больших массивов текстов. Международная алгебраическая конференция: К 100-летию со дня рождения П. Г. Конторовича и 70-летию Л. Н. Шеврина. Тез. докл. Екатеринбург: Изд-во Урал. ун-та, 2005, с. 173-175.

[2] Веретенников А. Б., Лукач Ю. С. Еще один способ индексации больших массивов текстов. Известия Уральского государственного университета. Серия «Компьютерные науки», 2006. №43. с. 103-122.

[3] Веретенников А. Б. Эффективное создание текстовых индексов. Проблемы теоретической и

прикладной математики: Труды 39-й Всероссийской молодежной конференции. Екатеринбург: УрО РАН, 2008. с. 348-350.
[4] Веретенников А. Б. Создание легко обновляемых текстовых индексов. Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды Десятой Всероссийской научной конференции «RCDL'2008». Дубна: ОИЯИ, 2008. с. 149-154.
[5] Веретенников А. Б. Библиотека для создания оконных интерфейсов на любых скриптовых языках в операционной системе Windows. Информационно-математические технологии в экономике, технике и образовании: Тезисы докладов Третьей международной научной конференции. Екатеринбург: УГТУ-УПИ, 2008. с. 220-221.
[6] Веретенников А.Б. Эффективная индексация текстовых документов с использованием CLB-деревьев. Системы управления и информационные технологии, 2009, 1.1(35). - С. 134-139.
[7] Веретенников А.Б. Гибкий подход к проблеме поиска похожих документов. Проблемы теоретической и прикладной математики: Труды 40-й Всероссийской молодежной конференции. Екатеринбург: УрО РАН, 2009. с. 392-396.
[8] Веретенников, А. Б. Сравнение эффективности CLB-дерева в 32-битных и 64-битных архитектурах. Материалы межвузовской научной конференции по проблемам информатики «СПИСОК 2009». Екатеринбург. 2009. с. 7-13.
[9] Веретенников А. Б. О методе оптимизации создания CLB-дерева. Проблемы теоретической и прикладной математики: Тезисы 41-й Всероссийской молодежной конференции. Екатеринбург: УрО РАН, 2010. с. 429-435.
[10] Веретенников А. Б. О платформе для электронной текстовой библиотеки. Материалы конференции «Математическое моделирование, численные методы и комплексы программ». Екатеринбург:Изд-во Урал. Ун-та, 2010. с.30-34.
[11] Веретенников А.Б. Теоретические исследования производительности CLB-деревьев. Системы управления и информационные технологии, 4.1(42), 2010. - С. 123-128.
[12] Веретенников А.Б. Метод организации индекса коллекции XML документов. Современные проблемы математики: тезисы 42-й Всероссийской молодежной школы-конференции. Екатеринбург: Институт математики и механики УрО РАН, 2011.
[13] Лукач Ю. С. Быстрый морфологический анализ флективных языков. Международная алгебраическая конференция: К 100-летию со дня рождения П. Г. Конторовича и 70-летию Л. Н. Шеврина. Тез. докл. Екатеринбург: Изд-во Урал. ун-та, 2005, с. 182-183.
[14] Сокирко А, Путрин А. Автоматическая обработка текста. www.aot.ru.
[15] Brown, E. W., Callan, J. P., and Croft, W. B. Fast incremental indexing for full-text information retrieval. In Proceedings of the International Conference on Very Large Databases, Santiago, Chile, 1994, pp. 192-202.
[16] Brown, E. W., Execution Performance Issues in Full-Text Information Retrieval. Technical Report 95-81. Computer Science Department University of Massachusetts at Amherst, 1995.
[17] Stefan Buttcher and Charles L. A. Clarke. A Hybrid Approach to Index Maintenance in Dynamic Text Retrieval Systems (2006). In ECIR 2006: Proceedings of the 28th European Conference on Information Retrieval, pp. 229-240.
[18] Heinz, Steffen, and Justin Zobel. Efficient single-pass index construction for text databases. *JASIST* 54(8), 2003, pp. 713-729.
[19] Lester, N., Zobel, J., Williams, H., Jan. In-place versus re-build versus re-merge: Index maintenance strategies for text retrieval systems. In: Estivill-Castro, V. (Ed.), Proc. Australasian Computer Science Conf., 2004, pp. 15.22.
[20] Nicholas Lester. Fast On-Line Index Construction by Geometric Partitioning. In CIKM ’05: Proceedings of the 14th ACM international conference on Information and knowledge management, 2005, pp. 776-783.
[21] Moffat, Alistair, and Justin Zobel. Self-indexing inverted files for fast text retrieval. TOIS 14(4), 1996, pp. 349-379.
[22] Prywes, N. S., Gray, H. J. The organization of a Multilist-type associative memory. IEEE Trans. on Communication and Electronics, 68 (1963), pp. 488-492.
[23] Shieh, W.-Y. and Chung, C.-P. A statistics-based approach to incrementally update inverted files. Inform. Proc. Manag. 41, 2, 2005, pp. 275-288.
[24] Tomasic A., Garcia-Molina H., and Shoens K. Incremental updates of inverted lists for text document retrieval. In Proc. ACMSIGMOD Int. Conf. on the Management of Data, Minneapolis, Minnesota, 1994, pp. 289-300.
[25] Zobel, Justin, and Alistair Moffat. Inverted files for text search engines. ACM Computing Surveys 38(2), 2006, Article 6.

Fulltext index for frequency updatable libraries

Veretennikov A.B.

Author introduce a new data structure for fast indexing of large volumes of texts and a software system based on this structure. The latter is more effective for inserting data into indexes than other known data structures, and is comparable with them in the data retrieval. An new approach for inserting small amount of new data given.