Математические вопросы кибернетики и вычислительной техники 24, 2005, 147-157.

Некоторые методы сжатия данных и их индексов в СУБД

Мигран С. Григорян

Институт проблем информатики и автоматизации НАН РА
e-mail mihran.grigoryan@buy.am

Аннотация

В данной статье рассматриваются некоторые методы сжатия табличных данных и индексных структур в СУБД. Приводятся сравнительные характеристики этих методов, и предлагаются их некоторые модификации.

1. Введение

Вопрос экономного кодирования информации в системах управления базами данных не потерял актуальности до сих пор. Многие современные исследователи отмечают недостаточную теоретическую проработку проблемы и неэффективность поддержки сжатия данных в промышленных СУБД.

Применение в СУБД экономного кодирования без потерь информации приводит к ряду положительных результатов. Наиболее очевидным эффектом является уменьшение физического размера базы данных, журнальных и архивных файлов. Но также часто достигается увеличение скорости выполнения запросов и снижение требований к объему оперативной памяти, что отмечается практически во всех работах в данной области. Поэтому эффективная реализация поддержки сжатия данных существенно улучшает качество СУБД.

Следует также отметить, что экономное кодирование способствует криптографической защите информации. Устранение статистической избыточности повышает криптостойкость алгоритмов и часто является предварительным действием в схемах шифрования данных.

Для применения в СУБД требуются специфические методы и приемы экономного кодирования, поскольку обычные методы не удовлетворяют ряду требований. Практическая ценность реализации достигается только при обеспечении быстрого доступа к произвольной записи или элементу данных. Востребованы так называемые методы сжатия с сохранением упорядоченности, позволяющие выполнять операции сравнения без декодирования данных.

В дальнейшем затронута только проблематика использования методов сжатия без потерь информации. Это объясняется следующим. Во-первых, применение экономного кодирования без потерь информации является более универсальным решением, чем сжатие с потерями. Во-вторых, эта область точнее отвечает интересам автора. В-третьих, приемы сжатия данных с потерями информации (в первую очередь мультимедийных данных) и способы их использования в СУБД обладают существенной спецификой и требуют отдельного рассмотрения.

В настоящей статье, предпринята попытка, систематически описать известные подходы к решению проблемы использования сжатия данных в СУБД, а также выявить недостаточно изученные и перспективные направления работ в проблемной области. Описанные методы и их модификации были применены в СУБД OCS разработанной автором данной статьи.

2. Основные методы сжатия данных. Терминология

Экономное кодирование информации достигается за счет представления маловероятных событий более длинными словами, чем событий с высокой вероятностью наступления. Если вероятность наступления события равна p, то, такое событие выгоднее кодировать словом длиною $\lceil \log_2 p \rceil$ битов. Методы сжатия данных явно или неявно опираются на этот факт.

В результате процесса экономного кодирования единице исходных данных (символу, слову, строке, числу и т.п.) ставится в соответствие так называемое кодовое слово. Кодовое слово состоит из последовательности цифр, обычно двоичных. Совокупность всех кодовых слов образует код. Если длины всех кодовых слов одинаковые, то используемый код имеет фиксированную (постоянную) длину, иначе — переменную. Если исходные данные могут быть однозначно восстановлены по массиву соответствующих кодовых слов, то кодирование не приводит к потере информации, т.е. является безущербным, без потерь.

Эффективность сжатия как характеристика сокращения размера представления информации относительно исходного будет в данном обзоре определяться степенью сжатия. Степень сжатия принимается равной отношению объема исходных данных к объему соответствующих им сжатых данных и измеряется указанным отношением.

Все методы сжатия принято разделять на два класса: методы статистического кодирования и методы словарного сжатия. В схемах сжатия также часто используются вспомогательные преобразования, обеспечивающие или способствующие выполнению этапа экономного кодирования.

2. 1. Статистическое кодирование

Методы статистического кодирования явным образом опираются на теорему Шеннона. Такие методы включают в себя два этапа: оценка вероятности кодируемых элементов (моделирование) и собственно кодирование. На этапе кодирования выполняется замещение элемента s_i с оценкой вероятности появления $q(s_i)$ кодовым словом длиной $-\log_2 q(s_i)$ битов. Этот этап иногда называется энтропийным кодированием. Оценки $q(s_i)$ могут быть получены из безусловных частот встречаемости элементов, условных частот, т.е. с учетом контекста, более сложными способами. Восстановление данных без потерь обеспечивается в том случае, когда кодер и декодер оперируют одними и теми же оценками $q(s_i)$ в каждый момент времени.

Задача кодирования элемента с заданной вероятностью традиционно решается с помощью разновидностей метода Хаффмана и арифметического сжатия [1].

Алгоритм Хаффмана определяет процедуру построения кода переменной длины, средняя избыточность которого минимальна для всех не блочных кодов, т.е. задающих отображение ровно одного исходного элемента в одно кодовое слово. Поскольку слово может быть представлено только целым числом битов, при кодировании по Хаффману осуществляется приближение $q(s_i)$ дробями, равными степеням двойки. Поэтому данный алгоритм неприменим непосредственным образом для экономного кодирования элементов бинарного алфавита. Код Хаффмана является префиксным и поэтому обычно представляется в виде дерева. Традиционно используется двухпроходная схема (статистический алгоритм Хаффмана): при первом просмотре данных подсчитывается статистика встречаемости элементов, т.е. строится модель данных, на основании которой формируется код; при втором просмотре данные сжимаются с помощью полученного кода. Оценки вероятности $q(s_i)$ при кодировании постоянны, и код не меняется.

Арифметическое сжатие, или арифметическое кодирование, позволяет при кодировании нескольких элементов представлять каждый элемент в среднем дробным числом битов. Поэтому обычно арифметическое сжатие обеспечивает большую степень сжатия, чем кодирование по Хаффману. Блок кодируемых элементов представляется дробью, определяемой произведением оценок вероятности $q(s_i)$ всех элементов блока. Это и определило название метода. Чем $q(s_i)$ меньше, тем длиннее дробь, и больше требуется двоичных символов для ее представления. Алгоритм декодирования сложнее, чем для метода Хаффмана, но позволяет восстановить исходные данные без потерь. Арифметическое кодирование не требует явной перестройки кода

при изменении оценок вероятности, поэтому обычно используется адаптивная однопроходная схема, позволяющая естественным образом учитывать локальные особенности данных.

2. 2. Словарное сжатие

Идея словарного сжатия заключается в замене последовательностей элементов исходных данных на идентификаторы таких фраз некоторого словаря, которые совпадают с замещаемой последовательностью. Методы словарного сжатия эксплуатируют факт повторяемости строк символов. Словарь как совокупность фраз может строиться различным образом. Например, в него могут включаться такие строки, которые обладают наибольшей величиной характеристики $q(L_i - L_j)$, где q — частота встречаемости последовательности, L — длина последовательности, L_j — длина идентификатора (указателя) фразы словаря.

Среди словарных схем, наибольшее распространение получили методы Зива-Лемпела. Словарные методы, относимые к этому классу, можно разделить на два семейства: LZ77 (LZ1) и LZ78 (LZ2). Схемы семейства LZ77 базируются на одноименном методе. В методах данного семейства роль словаря играет порция уже обработанных данных. Последовательность обычно кодируется указанием позиции начала эквивалентной фразы в словаре (смещения) и длины совпадения. Пара <смещение, длина совпадения> в этом случае является указателем. Если кодируемый элемент отсутствует в словаре, то он некоторым образом помечается и представляется, как есть. Такой элемент называется литералом. Из способа формирования словаря следует, что методы семейства LZ77 являются адаптивными.

На практике схемы типа LZ77 используются совместно с алгоритмами статистического кодирования указателей и литералов. Например, в методе LZH для экономного кодирования указателей и литералов используется алгоритм Хаффмана.

В схемах семейства LZ78 в словарь включаются не все последовательности, встреченные в обработанном массиве данных, а лишь "перспективные" с точки зрения вероятности появления в дальнейшем. Например, в методе LZ78 новая фраза формируется как сцепление (конкатенация) одной из фраз S словаря, имеющей самое длинное совпадение с текущей кодируемой последовательностью S', и символа s. Символ s является символом, следующим за последовательностью $S' = S$. В отличие от семейства LZ77, в словаре не может быть одинаковых фраз. В самом известном представителе семейства LZ78, методе LZW, словарь инициализируется фразами для всех символов алфавита кодируемых данных. Указатели фраз кодируются словами фиксированной длины, определяемой размером словаря. В рамках метода семейства LZ78 несложно реализовать эффективное неадаптивное и полу адаптивное сжатие, при котором словарь строится заранее.

Словарь, используемый в словарных методах сжатия, можно рассматривать как аналог статистической модели данных, применяемой в статистических методах.

2. 3. Преобразования, используемые в схемах сжатия данных

Сжатие данных может быть улучшено, если обрабатываются не исходные элементы, а их разности. Это характерно, например, для оцифрованных аналоговых сигналов, табличных данных. В последнем случае вычитаются не элементы, а последовательности элементов, т.е. строки таблицы. Для обеспечения возможности однозначного преобразования, необходимо сохранить в первоначальном виде первый элемент или любой другой, служащий опорным (базовым). Описанный прием называется дифференциальным или разностным кодированием.

При наличии локальных группировок одинаковых символов сжатие может быть улучшено за счет предварительного преобразования "стопка книг". Все используемые элементы заносятся в список и кодируются своими номерами в нем. При каждом появлении элемента он переносится в голову списка, "сдвигая" вниз все остальные. В результате локально часто используемые элементы представляются одинаковыми числами, что позволяет применять простые методы сжатия.

2.4. Классификация методов по стратегиям обновления модели (словаря)

Схемы сжатия данных классифицируются также по способу построения и обновления модели (словаря). Выделяют четыре варианта моделирования:

- статическое (неадаптивное);
- полуадаптивное;
- адаптивное (динамическое);
- блочно-адаптивное.

При статическом моделировании для любых обрабатываемых данных используется одна и та же модель (словарь). Иначе говоря, не производится адаптация к особенностям сжимаемых данных.

Полу адаптивное сжатие предполагает, что для кодирования заданной последовательности выбирается или строится модель на основании анализа именно обрабатываемых данных.

Адаптивное моделирование является естественной противоположностью статической стратегии. По мере кодирования модель изменяется по заданному алгоритму после сжатия каждого элемента. Однозначность декодирования достигается тем, что, во-первых, изначально кодер и декодер имеют идентичную модель и, во-вторых, модификация модели при сжатии и распаковке осуществляется одинаковым образом.

В случае блочно-адаптивной стратегии обновление модели может выполняться после обработки целого блока элементов, в общем случае переменной длины.

3. Сжатие табличных данных

Для табличных данных характерны следующие типы избыточности, которые обычно проявляются одновременно:

- разница в частоте встречаемости отдельных символов (элементов); например, если поле текстовое, то число пробелов обычно значительно превышает количество любых других символов;
- наличие последовательностей одинаковых символов; например, это могут быть серии отсутствующих значений или нулей, а также серии пробелов, дополняющие значения строкового типа до получения максимальной ширины, принятой для колонки; последний пример характерен для СУБД, не поддерживающих переменную длину значений строковых атрибутов;
- частое повторение определенных последовательностей символов;
- частое появление символов или последовательностей символов в определенных местах (позициях).

Для табличных данных характерны как вертикальные корреляционные связи (по столбцу), так и горизонтальные (по строке). Часто вертикальные зависимости сильнее горизонтальных. Лучшее сжатие будет обеспечиваться теми методами, которые эффективно эксплуатируют все типы избыточности.

Наиболее распространенными типами данных, используемыми в БД, являются строковые и числовые. Такие типы, как дата и время, могут быть представлены как числовые. Часто методы сжатия данных числовых и строковых (текстовых) типов имеют специфику, поэтому они рассмотрены отдельно.

3. 1. Кодирование числовых данных

Упаковка битов

Метод упаковки битов (bit packing) заключается в кодировании значения атрибута битовыми последовательностями фиксированной длины. Для каждого сжимаемого столбца требуется хранить соответствующую таблицу перекодировки. Разрядность кодовых последовательностей определяется количеством различных чисел, значения которых реально принимаются атрибутом, или же мощностью области определения атрибута. Очевидно, что требуемая длина кода N равна $\lceil \log_2 n \rceil$, где n — это мощность множества реальных или допустимых значений атрибута.

Упаковка битов — это простой способ сжатия на уровне значения атрибутов, реализация которого необременительна. Также важно, что при соответствующей реализации сохраняется упорядоченность. Но проблемой при использовании данного метода является обработка появления новых значений атрибута. В этом случае может потребоваться перекодировать все значения столбца, если $\lceil \log_2 n \rceil$ становится больше исходного N. В работах [5][6] предложена модификация метода для сжатия целых чисел, позволяющая в ряде случаев избежать перекодирования и естественным образом обеспечивающая сохранение упорядоченности. Алгоритм сжатия, названный автором "frame of reference compression" — "сжатие кадра ссылок", состоит из двух частей:

- уровень страницы — сжатие строк и их сохранение в отдельных страницах;
- файловый уровень — перераспределение данных между страницами с учетом способа сжатия уровня страницы для улучшения общей степени компрессии.

Данные, физически перераспределяемые при обработке на файловом уровне, кодируются при обработке на уровне страницы.

Суть метода: для каждого столбца находится минимальное и максимально значение. Значения, попавшие внутрь этого диапазона, последовательно кодируются. Длина кодового слова постоянна и определяется размером диапазона. Пример: пусть имеется последовательность строк из двух полей

$S = \{(100, 21), (98, 24), (103, 29)\}$.

Для первого поля диапазон [98, 103], для второго — [21, 29]. Мощность первого множества значений равна 6, второго — 9. Поэтому S будет представлена как:

$S = \{(010_2, 0000_2), (000_2, 0011_2), (101_2, 1000_2)\}$.

Очевидно, что если сжатие применяется к полям фиксированной длины, то их новая длина также фиксирована. Для каждого столбца необходимо хранить только границы диапазона и, возможно, длину. Перекодирование требуется лишь в случае выхода значения за границы диапазона.

Степень сжатия сильнее всего зависит от распределения значений каждого атрибута для записей, хранящихся в одной странице. Можно кардинально улучшить сжатие за счет перераспределения записей по страницам при обработке на файловом уровне.

В [5] предлагается применять алгоритм сжатия кадра ссылок и для экономного кодирования индексных структур древовидного вида. В частности, рассмотрено сжатие структур типа R-деревьев, в том числе с потерями информации внутри индекса. Последнее, позволяет достигать компромисса между используемым объемом внешней памяти, для хранения индекса, и скоростью поиска. Структура R-дерева такова, что сжатие с потерями не приводит к неоднозначности результатов поиска.

В [6] указывается, что предложенный алгоритм позволил сжать таблицы с данными о фондовых операциях в 3-5 раз. Скорость выполнения запросов при выборке данных в десятки раз превосходила вариант, при котором для сжатия использовался алгоритм типового компрессора GZip. Экономное кодирование R-деревьев в БД одной гео-информационной системы уменьшило размер их представления до 55% от исходного.

Метод упаковки битов не может обеспечить значительной степени сжатия относительно других методов, что, тем не менее, во многих приложениях может компенсировать высокой скоростью обработки данных и сохранением упорядоченности.

Кодирование длин серий и методы устранения констант

В общем случае, основной выигрыш при кодировании числовых данных может быть получен за счет сжатия ведущих нулей и экономного кодирования, часто повторяющихся значений, которыми обычно являются ноль и отсутствующее значение. Такие часто повторяющиеся значения называются в этом подпункте константами. Сжатие ведущих нулей часто выполняется с помощью упаковки битов, рассмотренной в предыдущем подпункте.

Последовательности констант при кодировании длин серий заменяются тройкой <флаг, константа, длина серии>. Существуют другие модификации метода, но общим недостатком является медленное декодирование, требующее $O(n)$ операций, где n — число элементов исходной строки.

Устранение определенной константы может быть реализовано с помощью битовой карты. При этом физически в БД сохраняются только не константные значения. Местоположение констант определяется битовой картой, каждый разряд которой равен, например, 1, если в соответствующей позиции находится обычное значение, и 0, если константа. Например, если исходная строка $S=\{D1,D2,c,D3,c,c,c,D4\}$, то сжатая строка $S'=\{D1,D2,D3,D4\}$ и битовая карта B={1,1,0,1,0,0,0,1}. Как и для кодирования длин серий, время декодирования равно $O(n)$.

В работах [2][3] была предложена модификация метода битовой карты, обеспечивающая в среднем более быстрый поиск в сжатых данных. В этом методе, названном "заголовочное сжатие", битовая карта преобразована в заголовочный вектор H, в нечетных позициях которого записывается число несжатых значений с накоплением, а в четных — число пропущенных констант с накоплением. Например, для исходной строки $S=\{D1,D2,D3,D4\}$ вектор $H=\{2,1,2+,1,+3,2+1+1\}=\{2,1,3,4,4\}$. Декодирование информации может быть выполнено посредством бинарного поиска по H, поэтому время декодирования пропорционально $O(\log|H|)$.

Используется так же, более сложная модификация метода битовой карты, позволяющая сократить число обращений к внешней памяти при декодировании. Метод назван "BAP", поскольку при сжатии используется три объекта: битовый вектор (bit vector, *BV*), адресный вектор (address vector, *AV*) и так называемый физический вектор (physical vector, *PV*). В позиции битового вектора записывается 0, если текущий символ равен константе, и 1, если наоборот. В физическом векторе перечисляются все не константные значения. Битовый вектор разбивается на подвектора, каждый из которых независимо сжимается с помощью кода Голомба и сохраняется в отдельном блоке или последовательности блоков устройства внешней памяти. В ячейке *AV(i)* адресного вектора хранится относительная позиция в физическом векторе последнего не константного элемента для подвектора с номером i-1. Первый элемент адресного вектора равен нулю. Если в предыдущем подвекторе одни константы, то в адресный вектор записывается значение его предыдущего элемента. Адресный вектор также может быть сжат с помощью дифференциального кодирования. При кодировании и декодировании на основании адресного вектора определяется, какой подвектор надо распаковать, и декодируется только он. Если адресный вектор достаточно мал, чтобы поместиться в оперативную память, то при декодировании требуется только одно обращение к внешней памяти.

Например, если размер подвектора равен 5, и константа есть "0", то для строки

S={1,0,0,4,0, 0,0,0,0,0, 0,0,8,0,0 0,12,0,0,0, 0,17,0,0,20}

содержимое векторов будет таким:

BV={1,0,0,1,0, 0,0,0,0,0, 0,1,0,0,0, 0,1,0,0,1},

PV={1,4,8,12,17,20},

AV={0,2,2,3,4}.

В частности, *AV(5)=4*, поскольку последний не константный элемент 4-го подвектора содержится в *PV(4)*. Это число 12.

Варьируя параметры, можно задавать соотношения между числом обращений, размером адресного вектора, размером блока, максимальной степенью сжатия и максимальным размером БД.

Следует подчеркнуть, что экономное кодирование констант актуально только для сильно разреженных БД. Например, такое свойство характерно для БД, в которых накапливаются данные о физических экспериментах.

Статистическое кодирование

Статистическое кодирование именно числовых данных редко упоминается в публикациях. Методы арифметического сжатия, кодирования по Хаффману и другие способы статистического кодирования применялись к данным произвольного типа. В [1] рассмотрена простая схема сжатия чисел посредством использования кода переменной длины, строящегося на основании статистики встречаемости чисел. Множество чисел разбивается на группы в соответствии с частотой использования так, что в группы малого размера попадают часто используемые значения. Элементы в пределах группы кодируются словами одинаковой длины $\lceil \log_2 n \rceil$, где n — размер группы. Кодовые слова разных групп различаются за счет использования флагов или префиксов. Такая схема обеспечивает в общем случае худшее сжатие, чем алгоритм Хаффмана, но может требовать меньше времени для кодирования.

Статистическое кодирование в силу достаточного затратного декодирования может иметь преимущество только при большой статистической избыточности числового массива и необходимости обеспечения высокой степени сжатия. Последнее требует применения сравнительно сложного моделирования, иначе будет более выгодным словарный подход к сжатию.

Методы Зива-Лемпела

Вопрос использования методов Зива-Лемпела для сжатия данных в СУБД исследован, в частности, в работах [8]. Авторы приходят к заключению, что методы Зива-Лемпела непрактичны для сжатия на уровне значения столбца. В [8] также указывается, что адаптивные методы проигрывают по степени сжатия неадаптивным (полу адаптивным).

Для использования в СУБД предложен модифицированный вариант метода Миллера-Уэгнама (Miller-Wegnam), или LZMW [1]. В этом методе каждая фраза словаря представляет собой конкатенацию двух других фраз, а не фразы и символа алфавита данных, что отличает метод от LZW. Сжатие в предложенной схеме выполняется на уровне строки и является полу адаптивным. Словарь строится при загрузке данных и может иметь размер до 4096 фраз. Указывается, что использование сжатия позволило сократить размер типовых БД в 2 раза при аналогичном увеличении скорости выполнения запросов. Расход тактов процессора увеличился всего на 20%. Данная схема была реализована в СУБД DB2 фирмы IBM.

Методы Зива-Лемпела часто использовались как эталонные при сравнении схем сжатия данных. Так, в [10] производится сравнение эффективности LZW с другими методами сжатия применительно к задаче экономного кодирования в СУБД. Аналогичную роль играет компрессор GZip, реализующий метод LZH, в [5][2].

Мы рассмотрели схему модификации метода LZW, позволяющая достичь свойства сохранения упорядоченности. Упорядоченность теряется при назначении разных кодовых слов строкам, которые являются префиксами друг друга. Поэтому для обеспечения сохранения порядка сортировки в словарь добавляются фразы, образуемые слиянием имеющихся фраз и "пустого" символа ("zilch"), не принадлежащего алфавиту данных. Эта модификация выполняется только для фраз, которые являются префиксами других фраз словаря. Указано, что предложенный прием применим к любым словарным схемам типа LZ78, задающим отображение исходной последовательности переменной длины в кодовое слово фиксированный длины, если все символы алфавита данных являются фразами словаря.

Основным недостатком методов Зива-Лемпела является сложность обеспечения произвольного доступа к информации. Для оценки жизнеспособности подхода было произведено его сравнение с другими словарными методами на данных теста TPC-H. В сопоставлении были задействованы реализации полуадаптивного алгоритма LZW, кодирования по словарю значений атрибута и кодирования по словарю из слов, встречающихся в значениях атрибута. Предложенный подход имел наибольшую эффективность среди рассмотренных методов, обеспечив двукратное сжатие при увеличении скорости последовательного доступа к данным на 30%.

4. Сжатие индексных структур

В настоящее время основными типами индексных структур, используемых в СУБД, являются B-деревья (B-tree) и битовые карты (bitmap), иначе называемые битовыми индексами [7]. Также применяются:

- проективные индексы (projection index) [7];
- R-деревья (R-tree) для многомерных данных [5];
- Датаиндексы (DataIndex) [7].

Число же экспериментальных схем равно, видимо, нескольким десяткам.

Наибольшее внимание в публикациях уделяется вопросу экономного кодирования битовых карт. Интерес объясняется тем, что данный тип индексов обычно позволяет выполнять запросы существенно быстрее, чем индексы других известных типов [7][9]. Но размер индексной структуры получается очень большим, если мощность множества значений индексируемых атрибутов велика.

В самом простом случае организации битовой карты каждой строке таблицы ставится в соответствие битовый вектор, состояние которого определяется значением индексируемого атрибута. *i*- ый бит принимает значение 1 для некоторой строки, если соответствующий атрибут этой строки имеет *i*-ое значение из множества всех значений атрибута. Например, если индексируется столбец "Пол", и область значений равна {мужской, женский}, то битовый вектор будет принимать значения:

- {1,0} для строк со значением атрибута "Пол" = "мужской";
- {0,1} для строк со значением атрибута "Пол" = "женский".
-

Только один бит вектора может быть единичным. Длина битового вектора равна $|A|$ битам, где $|A|$— мощность множества значений индексируемого атрибута.

Существуют другие разновидности битовых индексов [4][11]. Так называемые односторонние битовые индексы характеризуют результат одностороннего сравнения "<", а не точного "=". Битовые вектора в этом случае состоят из одной последовательности единиц и одной последовательности нулей. Также разработаны битовые индексы для двухстороннего сравнения. Битовые вектора, для одностороннего сравнения имеют длину $|A|-1$ битов, для двухстороннего — $\frac{|A|}{2}$ битов. Таким образом, проблема экономного представления индексов этих разновидностей также актуальна.

Сжатие битовых карт

В литературе обозначено два подхода к решению проблемы уменьшения размера битовых карт:

- декомпозиция, использование набора битовых карт [4][9];

- Сжатие собственно битовых карт [4][9][11].

Кроме уменьшения требуемого пространства для хранения, сжатие может обеспечить повышение скорости выполнения запросов. Это обуславливается следующим:

- уменьшается число операций обращения к внешней памяти;
- ряд булевых операций могут быть выполнений непосредственно над сжатыми данными, причем быстрее, чем над незакодированными.

В [9] произведено сравнение четырех алгоритмов сжатия битовых карт:

1. реализация LZH в библиотеке zlib, использующей тот же формат сжатых данных Deflate, что и компрессор GZip;
2. сжатие посредством zlib с предварительным кодированием длин серий (LZH+RLE);
3. код переменной длины, задаваемый алгоритмом ExpGol, два варианта;
4. код переменной длины, задаваемый алгоритмом байтового сжатия битовых карт (БСБК) — Byte-Aligned Bitmap Codes, два варианта.

Код ExpGol является расширением γ - кода Элайеса для представления целых чисел. Пусть V - это набор целых чисел v_i. При кодировании n необходимо найти такое k, что

$$\sum_{i=1}^{k-1} v_i < n \le \sum_{i=1}^{k} v_i$$

Пусть

$$d = n - 1 - \sum_{i=1}^{k-1} v_i$$

Тогда кодовое слово для n состоит из кодового слова для k и числа d, записанного с помощью $\lceil \log_2 d \rceil$ битов. Для представления k может быть использован тот же γ - код Элайеса: $\lfloor \log_2 k \rfloor$ нулевых битов, за которыми следует единичный бит. Например, числам {1,2,3,4} при использовании γ - кода соответствуют слова {1,010, 011, 00100}. Строго говоря, в соответствии с обозначениями в исходной статье Элайеса это не γ - код, а γ' - код (гамма-штрих код). Для эффективного кодирования последовательности нулей используется

$V = \{1b, 2b, 4b, 8b, 16b, \ldots\}$.

Параметр b выбирается равным медиане распределения длин серий нулей для первого варианта кода ExpGol и геометрического среднему — для второго.

Код БСБК обеспечивает высокую скорость кодирования-декодирования, поскольку все действия производятся над последовательностями байтов. Булевы операции над битовыми индексами, закодированными посредством БСБК, могут выполняться существенно быстрее, чем над незакодированными.

Код БСБК может быть односторонним и двухсторонним. Односторонний код позволяет кодировать серии нулей, а двухсторонний — как нулей, так и единиц. Каждое кодовое слово одностороннего кода состоит из двух частей:

"окно" (gap);
окончание (ending).

Содержимое "окна" определяет, сколько нулевых байтов предшествует окончанию. Окончание может содержать последовательность незакодированных байтов или специальный контрольный байт. Используется ряд приемов для экономного кодирования размеров "окна" и окончания. Таким образом, БСБК есть разновидность байт-ориентированного кодирования длин серий.

В [9] указывается, что алгоритм LZH+RLE не продемонстрировал ни разу лучшего сжатия, поэтому его характеристики не приводятся. Было установлено, что степень сжатия ExpGol для

двух вышеописанных вариантов выбора b практически не отличается. Поэтому характеристики варианта кода при вычислении b по геометрическому среднему также были опущены. В сравнении были использованы как односторонний код БСБК, так и двухсторонний. Из приводимых результатов сравнений для сгенерированных и реальных индексных данных следует:

- наибольшая степень сжатия для "густых" битовых карт с относительно большим количеством единиц обеспечивается LZH; в случае кластеризации серий нулей и единиц преимущество может иметь двухсторонний БСБК;
- наибольшую степень сжатия для разреженных битовых карт с вероятностью появления единицы $p < 0.1$ дает ExpGol; например, при $p = 0.0001$ степень сжатия для ExpGol в три раза выше, чем у LZH;
- большую скорость выполнения операций с индексами для "густых" битовых карт обеспечивает LZH;
- наибольшую скорость выполнения операций с индексами для разреженных битовых карт дает использование БСБК.

Необходимо отметить, что формат Deflate, в соответствии с которым реализована библиотека сжатия zlib, налагает ограничение на длину кодируемой за один шаг строки (длину совпадения) [1]. Этот порог равен 258 битам, что в данных условиях является жестким ограничением и неизбежно должно ухудшать степень сжатия сильно разреженных битовых карт.

В работе рассмотрен ряд проблем использования битовых индексов для поддержки сложных многомерных запросов к научным данным, содержащим большое количество не дискретных атрибутов, в первую очередь чисел с плавающей запятой. При этом изучены вопросы сжатия соответствующих модификаций битовых карт. В частности, применительно к решаемой задаче исследован вариант двухстороннего БСБК. Приводятся результаты ряда экспериментов, указывающие на ускорение выполнения запросов некоторых типов при использовании сжатых индексов.

В работе предложена схема сжатия на основе кодирования длин серий, обеспечивающая большую скорость выполнения операций с индексами, чем БСБК. Метод назван авторами "пословное гибридное кодирование длин серий" (Word-Aligned Hybrid run-length code, WAH). Название определено тем, что закодированные данные группируются в машинные слова, а не в байты, как в БСБК. Соответственно, вся обработка может выполняться на уровне слов. Определены два типа слов: заполнители ("fill words") и литералы. Слова различаются значением старшего бита. В слове-заполнителе указывается значение бита, образующего серию, и длина серии таких одинаковых значений. В литералах сохраняются в исходном виде те биты, которые не удалось сжать с помощью заполнителей. Требуется, чтобы длина серии, кодируемой заполнителем, была кратной числу рабочих битов литерала. Например, если машинное слово состоит из 32 битов, то длины совпадений должны быть кратны 31.

Результаты сравнительных экспериментов, свидетельствуют о 12-кратном превосходстве в скорости выполнения запросов при использовании предложенной схемы пословного кодирования относительно сжатия с помощью БСБК. Размер индексной структуры при этом на 60% больше размера структуры для БСБК. Для тестирования была использована таблица с данными результатов физических экспериментов, содержащая порядка 2.2 миллиона записей. Отмечается, что при вероятности появления в битовой карте единицы p=0.01 … 0.001размер индекса для предложенного метода примерно в 2.5 раза превышает размер индекса для БСБК.

Следует отметить, что основной проблемой всех способов сжатия битовых индексов, основанных на кодировании длин серий, является решение задачи эффективного декодирования с произвольным доступом.

Экономное кодирование B-деревьев

Сжатие индексных данных в B-дереве позволяет не только уменьшить размер структуры, но и приводит к получению более "плоского" дерева, т.е. с меньшей высотой. Это ускоряет доступ к отдельному ключу.

Обеспечение возможности непосредственного сравнения сжатых данных требует использования алгоритмов кодирования с сохранением упорядоченности. Для сжатия вершин B-

дерева может быть использован предложенный вариант метода упаковки битов, названный "сжатие кадра ссылок". Эта модификация была рассмотрена выше в подпункте "Упаковка битов" пункта "Кодирование числовых данных". Как и обычный метод упаковки битов, данный вариант сохраняет порядок сортировки, но годен только для кодирования целых чисел. Алгоритм является словарным и в этом сходен с предложенной схемой модификации LZW для сохранения упорядоченности. Но он обеспечивает большую степень сжатия. В статье указано, что предложенный алгоритм позволил сжать реальные индексные данные в 5 раз.

Литература

[1] Витолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. – М.: ДИАЛОГ-МИФИ, 2002. – 384 с.

[2] Alsberg P. A. Space and Time Savings Through Large Data Base Compression and Dynamic Restructuring. *Proc. IEEE* 63(8):1114-1122, August 1975.

[3] Buchsbaum A. L., Caldwell D. F., Church K. W., Fowler G. S., and Muthukrishnan S. Engineering the compression of massive tables: an experimental approach. *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 175-184, 2000.

[4] Cannane A., Williams H. E., and Zobel J. A General-Purpose Compression Scheme for Databases. *Proc. IEEE Data Compression Conference*, p. 519, 1999.

[5] Chan C.Y. and Ioannidis Y.E. An Efficient Bitmap Encoding Scheme for Selection Queries. *Proc. ACM SIGMOD Intl' Conference*, Philadelphia, Pennsylvania, June 1999, pp. 215-226.

[6] Goldstein J. Improved query processing and data representation techniques. A dissertation submitted in partial fulfillment of the requirements for the degree of doctor of philosophy (computer sciences) at the University of Wisconsin – Madison. 1999.

[7] Goldstein J., Ramakrishnan R., and Shaft U.. Compressing relations and indexes. *Proc. IEEE Conf. on Data Engineering*, Orlando, FL, USA, pp. 370-379, 1998.

[8] Goyal K., Ramamritham K., Datta A., Thomas H. Indexing and Compression in Data Warehouses. *Technical Report, Indian Institute of Technology, Bombay*, April 1999.

[9] Iyer B. R. and Wilhite D. Data Compression Support in Databases. *In Proceedings of the 20th International Conference on Very Large Data Bases*, Santiago, Chile, pp. 695-704. 1994.

[10] Johnson T. Performance Measurements of Compressed Bitmap Indices. *Proceedings of 25th International Conference on Very Large Data Bases*, September 7-10, 1999 (VLDB'99), Edinburgh, Scotland, UK, pp. 278-289.

[11] MySQL AB (2004). MySQL Reference Manual for version 4.0.18.

[12] Stockinger K. Multi-Dimensional Bitmap Indices for Optimising Data Access within Object Oriented Databases at CERN. *PhD*. Nov. 2001.

ՏՀՂՀ – ում տվյալների և ինդեքսների սեխման մի քանի ալգորիթմ

Մ. Գրիգորյան

Ամփոփում

Այս հոդվածում դիտարկվում են մի շարք աղյուսակային տվյալների սեղման մեթոդներ: Տրվում են նրանց համեմատական բնութագրերը և բերվում են մի շարք կատարելագործումներ: