Сжатие данных в хранилище больших графов*

И. В. ПОЛЯКОВ
Национальный исследовательский университет «Высшая школа экономики»
e-mail: igorp86@mail.ru

А. А. ЧЕПОВСКИЙ
Национальный исследовательский университет «Высшая школа экономики»
e-mail: c4hapa@gmail.com

А. М. ЧЕПОВСКИЙ
Национальный исследовательский университет «Высшая школа экономики»
e-mail: achepovskiy@hse.ru

УДК 004.421.2:519.178

Ключевые слова: хранилище графов, сжатие данных, большие данные.

Аннотация

В статье рассматриваются методы сжатия данных для хранения графов больших размеров. Предлагаются алгоритмы препроцессинга графа специальной структуры для повышения плотности записи данных и повышения эффективности выполнения базовых операций с графами.

Abstract

I. V. Polyakov, A. A. Chepovskiy, A. M. Chepovskiy, Data compression in big graph warehouse, Fundamentalnaya i prikladnaya matematika, vol. 21 (2016), no. 4, pp. 125—132.

In this paper, we propose an approach for compact storage of big graphs. We propose preprocessing algorithms for graphs of a certain type, which can significantly increase the data density on the disk and increase performance for basic operations with graphs.

1. Введение

Специализированные хранилища графов получили широкое распространение в последние годы. Они имеют обширную область применения от химических и биологических задач до задач хранения и обработки графов социальных сетей. Их преимуществом по сравнению с реляционными базами данных является поддержка более сложных операций, необходимых для моделей исследуемых предметных областей. Каждая такая база данных имеет свои особенности и

*Работа выполнена при поддержке РФФИ, гранты № 16-29-09546 офи_м и 16-07-00641.

свой язык запросов. Сравнительную характеристику наиболее популярных хранилищ графов можно найти в [3—5, 11].

Сжатие данных в базах общего назначения обычно выполняется одним из стандартных механизмов, как это происходит в GBASE [8]. В то же время для графов с определённой структурой атрибутов связей можно предложить более эффективные методы сжатия данных, позволяющие более чем в два раза повысить плотность хранения данных. Это достигается за счёт более детального учёта специфики возникновения данных. В целом алгоритмы, учитывающие специфику сжимаемых данных, способны демонстрировать значительно более высокие показатели степени сжатия, чем распространённые алгоритмы общего назначения. Одним из таких алгоритмов является PAQ [9, 10], демонстрирующий высокие показатели степени сжатия в различных тестах. Недостатком данного алгоритма является крайне низкое быстродействие и высокие требования к объёму потребляемой памяти. Показатели, достигаемые алгоритмами сжатия, существенно зависят от размера порции сжимаемых данных [6]. Сжатие данных небольшими порциями приводит к низкой скорости сжатия и невысокой степени компрессии.

Размер рассматриваемых графов не позволяет полностью разместить их в оперативной памяти компьютера, что налагает определённые требования на реализацию аналитических операций над ними. Первостепенную роль приобретает операция выгрузки подграфа, задаваемого окрестностью некоторого множества вершин, для последующей визуализации и детального анализа. Такие операции, как поиск вершин по какому-либо набору атрибутов, а также поиск путей между двумя множествами вершин могут быть успешно реализованы на полном графе, (см. [2]). Для ускорения операций выгрузки подграфов и поиска путей в [1] используются специализированные структуры данных, оптимизирующие число операций с диском во время их выполнения.

Мы рассматриваем произвольные графы больших размеров, описывающие взаимодействие объектов. Предполагается, что связи описываются простой структурой атрибутов (например, только временем взаимодействия и типом связи). В таком случае становится возможным обеспечивать компактное хранение графа с использованием специальных алгоритмов сжатия. Предполагается, что для двух объектов допустимо объединять несколько имевших место взаимодействий между ними в одно. Тогда возникает метаграф, содержащий информацию о том, имело ли место хотя бы одно взаимодействие между двумя произвольными объектами. Объединение не обязательно должно производиться на всём интервале атрибута связи (например, за весь временной промежуток, для которого построен граф). Допустимо объединение взаимодействий в рамках заранее заданных интервалов значений конкретного атрибута связи. В предлагаемом методе решаются задачи снижения объёма дискового пространства, необходимого для хранения графа, размещение которого в оперативной памяти невозможно.

2. Используемые определения

Множества вершин и связей хранимого графа G обозначим через $V(G)$, $E(G)$ соответственно. Для каждой связи e из множества $E(G)$ определены в качестве атрибута функции $T(e)$ (например, соответствующая времени взаимодействия объектов, которые соединены данной связью), $C(e)$ — тип взаимодействия. Множество допустимых типов взаимодействия обозначим через S. Пусть T_0 — начальное значение функции взаимодействия. Таким образом, граф содержит взаимодействия объектов, которые задаются на некотором отрезке $[T_0, T]$, где T — текущее значение атрибута связи.

Для атрибутов вершин определён набор доменов $A_1, A_2, \ldots, A_D$. Каждая вершина v из множества $V(G)$ имеет один или несколько атрибутов, принимающих значения из доменов $A_1, A_2, \ldots, A_D$. Один из атрибутов — $k(v)$ — считается ключевым, его значение должно уникальным образом идентифицировать вершину. Остальные атрибуты вершины обозначим через $a_1(v), \ldots, a_{N(v)}(v)$. Каждый из них принимает значение в одном из доменов $A_1, A_2, \ldots, A_D$. Допускается, чтобы два и более атрибутов принимали значения в одном и том же домене.

На всех доменах считается определённой функция хэширования HASH на множество натуральных чисел и ноль:

$$\text{HASH}\colon A_i \to \{0, 1, \ldots\}.$$

Используя значение, получаемое для ключевого атрибута, можно считать функцию HASH определённой на множестве вершин:

$$\text{HASH}(v) = \text{HASH}\big(k(v)\big).$$

Если в качестве $k(v)$ выступает некоторый числовой идентификатор, то в качестве хэширующей функции может быть взято тождественное отображение, поскольку данный идентификатор уже задан на домене натуральных чисел.

3. Схема хранения связей

Множество вершин V разбивается на M кластеров $V_0, V_1, \ldots, V_{M-1}$ по значениям функции HASH:

$$V_i = \{v \in V\colon \text{HASH}(v) \equiv i \pmod M\}.$$

Хранение связей кластера V_i на диске выполняется совместно в виде списка блоков

$$L_i = \{B_j^i\}, \ldots j = 1, \ldots, N_i,$$

объединённых в двунаправленный список. Каждый блок содержит набор записей, содержащих следующие поля:

$k(v_1^m)$ — ключевой атрибут первой вершины связи,
$k(v_2^m)$ — ключевой атрибут второй вершины связи,

T^m — атрибут связи (например, время взаимодействия),

C^m — тип связи,

при этом вершина v_1^m обязана принадлежать кластеру V_i.

Хранение блоков списка L_i выполняется внутри некоторого файла системы NTFS. Все блоки лежат в интервале адресного пространства файла заранее определённого размера. Каждый блок B_j^i состоит из следующих элементов:

$$B_j^i = \left(R(i,j), L(i,j), \mathrm{Mask}_j^i, t_{\min}^{ij}, t_{\max}^{ij}, \bigcup_{l=1,\ldots,N_{ij}} C_j^{il} \right),$$

где

$R(i,j)$ — указатель на предыдущий блок списка,

$L(i,j)$ — указатель на следующий блок списка,

Mask_j^i — битовая маска фиксированного размера N_M, при добавлении очередной связи устанавливается бит с номером $\mathrm{HASH}(v_1^m) \pmod{N_M}$,

$t_{\min}^{ij}$, $t_{\max}^{ij}$ — границы интервала значений атрибута, содержащие минимальное и максимальное значение атрибута по всем вошедшим в данный блок записям.

Хранение элементов $R(i,j)$, $L(i,j)$, Mask_j^i, $t_{\min}^{ij}$, $t_{\max}^{ij}$ выполняется в оперативной памяти компьютера. Битовая маска Mask_j^i используется для быстрой фильтрации блоков в том случае, если нужно получить все связи заданной вершины v. В таком случае можно пропустить блоки, в битовой маске которых не установлен бит $\mathrm{HASH}(v) \pmod{N_M}$. Каждый подраздел C_j^{il} формируется по мере добавления связей, относящихся к вершинам кластера V_i. Каждый кластер получает в оперативной памяти буфер фиксированного размера. Добавляемые связи, относящиеся к вершинам кластера V_i, хранятся в буфере. По мере заполнения буфера выполняется его сжатие алгоритмом, описанным в следующем разделе. На основе сжатых данных происходит формирование очередного подраздела C_j^{il}. При этом к сжатым данным дописываются поля $t_{\min}^{ijl}$, $t_{\max}^{ijl}$, определяемые как минимальное и максимальное значение атрибута по всем связям, вошедшим в данный буфер. Сформированный подраздел должен быть добавлен в последний блок двунаправленного списка. Если в последнем блоке недостаточно свободного места, происходит выделение очередного блока, который добавляется в конец списка L_i. При добавлении очередного подраздела в блоке производится обновление элементов Mask_j^i, $t_{\min}^{ij}$, $t_{\max}^{ij}$.

4. Алгоритм сжатия данных

На момент переполнения буфер состоит из массива записей вида

$$(k(v_1^m), k(v_2^m), T^m, C^m), \quad m = 1, \ldots N_b.$$

Алгоритм сжатия основан на нескольких утверждениях, выполняющихся для данных в буфере.

1. Разброс значений атрибута T^m по всем записям оказывается незначительным, за исключением небольшого числа выбросов.
2. Вектор $\{C^m\}$ часто оказывается постоянным в рамках записей, имеющих одно и тоже значение $k(v_1^m)$, один из возможных типов взаимодействий встречается на порядок чаще остальных.
3. Вектор $\{k(v_1^m)\}$ содержит небольшое число уникальных значений. Зачастую данные о связях заполняются порциями, содержащими большое количество связей одной и той же вершины, что приводит к появлению большого числа блоков буфера, заполненных связями только одной вершины.
4. Порядок, в котором записи расположены в буфере, несуществен и может быть утерян при выполнении операций компрессии и декомпрессии.

Для оптимального сжатия данных с учётом вышеперечисленных закономерностей выполняется предобработка данных сжимаемого буфера, после чего сжатие выполняется одним из стандартных алгоритмов, например методом DEFLATE [7]. Для лучшей совместимости со стандартными алгоритмами сжатия итоговое кодирование знаковых чисел использует младший бит числа для хранения его знака и остальные биты для хранения его модуля. Такой формат подходит лучше дополнительного кода, поскольку приводит к наличию большего числа нулевых байтов в коде небольших отрицательных чисел.

Алгоритм предобработки данных состоит из нескольких шагов.

Шаг 1. Определяется медиана T_M вектора $\{T^m\}$, $m = 1, \ldots, N_b$, после чего из каждой компоненты вектора вычитается T_M:

$$T^m = T^m - T_M.$$

Шаг 2. Производится лексикографическая сортировка записей по полям $k(v_1^m)$, $k(v_2^m)$, C^m, T^m. Записи, имеющие одно и то же значение $k(v_1^m)$, объединяются в одну группу. Первая запись каждой группы получает дополнительное поле со значением, равным числу записей в группе, из остальных записей удаляется поле $k(v_1^m)$, так как оно уже хранится в первой записи. В первую запись также добавляется целочисленное поле, равное длине текущей группы.

Шаг 3. Множество записей каждой группы, полученной на предыдущем шаге, разбивается на подгруппы по значению поля $k(v_2^m)$, которое сохраняется только в первой записи каждой подгруппы. Первая запись каждой подгруппы, кроме последней, получает дополнительное целочисленное поле со значением, равным длине текущей подгруппы. Отметим, что длину последней подгруппы хранить не нужно, так как её можно вычислить, зная длину всей группы и длины всех её подгрупп, кроме последней. Таким образом, хранится только вектор $(L, L_1, \ldots, L_{G-1})$, содержащий длины всей группы и всех подгрупп, кроме последней.

Шаг 4. Проверка условия, что «все записи некоторой подгруппы имеют одно и то же значение поля C^m». В каждой подгруппе, где оно выполнено,

поле C^m остаётся только в первой записи группы. При этом подгруппа снабжается специальным битовым флагом, отражающим результат проверки этого условия, который также помещается в первую запись.

Шаг 5. Производится разбиение каждой подгруппы, полученной на шаге 3 алгоритма, по значению поля C^m. Подгруппы, прошедшие проверку на шаге 4, разбиваться не будут. Значение поля C^m остаётся только в первой записи каждой подгруппы, к которой также добавляется поле, содержащее длину подгруппы.

Шаг 6. Производится преобразование полей T^m для каждой подгруппы, полученной на предыдущем шаге. Из значения данного поля для текущей записи подгруппы, начиная со второй, вычитается его значение для предыдущей записи:

$$D^m = T^m - T^{m-1}.$$

Данное преобразование обратимо и приводит к получению на каждой позиции небольшого неотрицательного целого числа.

Все шаги полученного алгоритма, за исключением второго, обратимы. Исходный порядок записей не может быть восстановлен, однако сами записи могут быть восстановлены путём обращения действия выполненных на каждом шаге алгоритма.

В результате работы с реальными данными дополнительно выявлено наличие нескольких часто встречающихся паттернов, учёт которых способен дополнительно повысить эффективность предобработки данных.

1. Многие группы по первому полю $k(v_1^m)$, имеющие небольшую длину, разбиваются на подгруппы, состоящие ровно из одного элемента.
2. Большинство подгрупп по полям $k(v_1^m)$, $k(v_2^m)$ имеют совпадающие значения поля C^m.
3. Для больших подгрупп по полям $k(v_1^m)$, $k(v_2^m)$, C^m наблюдается некоторая регулярность в полях D^m. Интервал времени между двумя взаимодействиями может совпадать. Учёт такого паттерна выполняется путём определения среднего значения между двумя последовательными взаимодействиями: D_{avg}. Затем к сжимаемым данным добавляется значение D_{avg}, а поля D^m заменяются на $D^m - D_{\text{avg}}$.

Проверка на наличие каждого паттерна приводит к появлению лишнего битового флага в сжимаемых данных и позволяет не хранить значительный объём информации, в случае если проверка оказалась успешной.

5. Заключение

Эффективность предложенной структуры подтверждена экспериментами с данными различных типов, которые показали, что предложенный алгоритм

даёт более чем одиннадцатикратную степень сжатия по сравнению с первоначальным объёмом буфера. Отметим, что использование стандартных алгоритмов (например, DEFLATE [7]) даёт меньшее сжатие, допускающее кратность порядка 6. При этом используемый алгоритм препроцессинга требует незначительного количества вычислительных ресурсов и практически не замедляет работу системы. Достигаемая степень сжатия позволяет существенно повысить плотность хранения данных на дисковых накопителях. Предложенный алгоритм позволяет достигать более чем двукратного снижения числа дисковых операций, необходимых для получения связей заданной вершины. Такой результат достигается тем, что при прочих равных условиях списки L_i состоят из меньшего числа блоков. Разработанный метод предоставляет возможность поместить все данные некоторого графа на одном узле локальной сети, что приводит к резкому росту эффективности операций с графами, которые используют поиск в ширину за счёт уменьшения дорогостоящих обменов данными между узлами локальной сети. Предложенная в данной работе модель хранения и сжатия связей позволяет существенно повысить степень сжатия при приемлемом быстродействии за счёт использования предобработки данных, учитывающей структуру сжимаемой информации. Хранение данных организовывается таким образом, чтобы обеспечить как можно больший объём порции сжимаемых данных, давая тем самым дополнительный выигрыш в быстродействии и степени сжатия. Разработанный метод позволяет полностью размещать в оперативной памяти специальные графы большего размера и обеспечивает высокую скорость выполнения операций, основанных на поиске в ширину или глубину. Данное свойство алгоритма существенно при работе с графами очень большого объёма.

Литература

[1] Поляков И. В., Чеповский А. А., Чеповский А. М. Хранение и обработка графа социальных сетей // Вестн. НГУ. Сер. Информ. технол. — 2013. — Т. 11, № 4. — С. 77—83.

[2] Поляков И. В., Чеповский А. А., Чеповский А. М. Алгоритмы поиска путей на графах большого размера // Фундамент. и прикл. матем. — 2014. — Т. 19, вып. 1. — С. 165—172.

[3] Angles R. A. Comparison of current graph database models // Proc. of the 2012 IEEE 28th Int. Conf. on Data Engineering Workshops, ICDEW '12. — Washington: IEEE Comput. Soc., 2012. — P. 171—177.

[4] Angles R. A., Gutierrez C. Survey of graph database models // ACM Comput. Surv. — 2008. — Vol. 40, no. 1. — P. 1—39.

[5] Batra S., Tyagi C. Comparative analysis of relational and graph databases // Int. J. Soft Comput. Eng. — 2012. — P. 509—512.

[6] Bell T., Witten I. H., Cleary J. G. Modeling for Text Compression // ACM Comp. Surv. — 1989. — Vol. 21, No. 4. — P. 557—591.

[7] Deorowicz S. Universal lossless data compression algorithms: Doctor of Philosophy Dissertation. — Gliwice: Silesian Univ. of Technology, Faculty of Automatic Control, Electronics and Computer Science, Institute of Comput. Sci., 2003.

[8] Kang U., Tong H., Sun J., Lin C.-Y., Faloutsos C. GBASE: a scalable and general graph management system // Proc. of the 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, August 21—24, San Diego, USA (2011).

[9] Mahoney M. Adaptive Weighing of Context Models for Lossless Data Compression: Tech. Report CS-2005-16. — Melbourne: Florida Inst. of Technology, CS Department, 2005.

[10] Peter P., Hoffmann S., Nedwed F., Hoeltgen L., Weickert J. Evaluating the true potential of diffusion-based inpainting in a compression context // Signal Processing: Image Communication. — 2016. — No. 46. — P. 40—53.

[11] Shrinivas S. G. Applications of graph theory in computer science: an overview // Int. J. Eng. Sci. Tech. — 2010. — No. 9. — P. 4610—4621.