Метод доступа для эффективного хранения сверхдлинных скалярных объектов

Е.Г. Владимирова
Санкт-Петербургский университет
198904 С.-Петербург, Библиотечная пл. 2
E-mail: ban@hq.math.lgu.spb.su

Аннотация

Предлагаемая работа описывает метод хранения больших бинарных объектов, состоящих из скалярных значений переменной длины. Система хранения не накладывает ограничений на размер объекта. Рост объекта возможен в любом направлении.

Далее описываются структуры данных и алгоритмы, реализующие предлагаемый метод хранения.

1 Введение

Одним из важных направлений развития баз данных являются разработки в области геоинформационных систем, систем компьютерного проектирования, экспертных и издательских системы и т.д. Многие из создаваемых в настоящее время систем баз данных основываются на объектно-ориентированной модели. При этом хранимые в них объекты могут достигать произвольно большого размера. Книги, оцифрованные географические карты, чертежи, видео- и аудиозаписи являются лишь несколькими примерами больших объектов. Эффективное управление большими объектами также необходимо во всех объектно- ориентированных и расширенных реляционных системах управления базами данных. Независимо от конкретного приложения, во всех таких системах необходима низкоуровневая поддержка стандартных конструкций данных, таких, как длинные списки или массивы.

Управление большими объектами накладывает определенные требования на систему хранения базы данных. В идеале система хранения должна быть создана таким образом, чтобы поддерживать объекты виртуально неограниченного размера (насколько позволяет физический объем внешней памяти).

2 Анализ родственных систем

Наибольшую известность получили три системы хранения, управляющие большими объектами.

Метод хранения объектов произвольной величины в системе EXODUS [1], разработанный в Висконсинском университете, позволяет произвольно измененять размеры объектов, причем увеличение или уменьшение размера одной части объекта не затрагивает других его частей. EXODUS обеспечивает удаление и вставку новых частей объекта в любом месте этого объекта. Объект в этом случае рассматривается как последовательность байтов, а верхние уровни абстракции как записи и индексы поддерживаются отдельно от объекта. Физически база данных хранит указатели на каждый объект хранения. Если объект занимает не более одной дисковой страницы, указатель ссылается непосредственно на страницу данных. Большие объекты представляются в виде последовательности сегментов данных. Размер сегмента может быть определен пользователем как целое число дисковых страниц. Идея заключается в том, что для часто изменяемых объектов сегмент данных лучше задавать размером одной дисковой страницы, а для достаточтно статичных объектов удобнее использовать большие сегменты данных. Сегменты данных индексируются структурой, близкой к структуре B- дерева, где ключами являются "порции"хранимого объекта. Данная система является низкоуровневой расширяемой системой, которая легко позволяет использовать ее как основу для любого приложения.

Метод хранения больших объектов, предлагаемый системой Starburst [6], делит большой объект на сегменты данных, где размер каждого следующего сегмента определяется размером предыдущего. Если размер большого объекта не известен заранее, каждый следующий сегмент удваивает в размере предыдущий, до тех пор, пока не дости-

гается максимальный возможный размер сегмента, затем резервируется последовательность сегментов максимального размера, пока длинное поле не умещается целиком. Если же размер большого объекта заранее определен, то используются сегменты максимальной длины. В любом случае, последний сегмент упорядочивается, т.е. неиспользовнные блоки с правого конца освобождаются. На большой объект заводится описатель, который содержит размер первого и последнего сегмента и массив указателей на зарезервированные сегменты большого объекта. Размер промежуточных сегментов неявно задается размером первого сегмента и известным характером роста. Эта схема может эффективно поддерживать последовательное и произвольное чтение, добавление и побайтовое замещение, но она не расчитана на вставку и удаление из середины объекта, так как эти операции обязательно изменяют размер большого объекта, и весь объект или его большая часть должны быть скопированы в новые сегменты. Представляется наиболее удобным использовать метод хранения системы Starburst для приложений, манипулирующих объектами, предназначенных для чтения (например, видео- и аудиозаписи), которые не предполагают частых модификаций, связанных с изменением длины.

Наиболее интересным представляется механизм хранения больших объектов, предлагаемый системой EOS [7], который объединяет две идеи, использованные в EXODUS и Starburst. Это можно рассматривать как генерацию двух структур, без ограничения на то, что сегменты, составляющие большой объект, должны быть фиксированной длины, или изменять размер, следуя некоторым предпопределенным шаблонам, как в Starburst. Каждый сегмент переменной длины составлен из физически последовательных дисковых страниц. Эти сегменты затем индексируются стуктурой, внутренние узлы которого аналогичны EXODUS. При частом использовании операций вставки и удаления сегменты могут разбиваться на более мелкие, что представляется невыгодным при такой организации данных, так как борьба с мелкими сегментами достаточно трудоемка.

Большинство существующих приложений использует для хранения объектов описанные методы. Все эти методы рассматривают объект как неделимую последовательность байтов и поддерживают следующие операции, которые манипулируют определенным числом байтов внутри объекта:

- считать и заменить некоторое число байтов внутри объекта,
- вставить и удалить байты внутри объекта,
- добавить байты в конец объекта.

Представляется не удобным интерпретировать объект как неделимую последовательность байтов для использования в приложении.

Система хранения для одной из наиболее известных объектно- ориентированных систем O_2 [2] опирается на Висконсинскую систему хранения. Объекты хранения в системе O_2 могут быть трех типов: кортежи, списки и наборы. Большой объект хранится как неделимая последовательность байтов, разбитых на блоки данных. Индексное оглавление к блокам данных построено в виде B-дерева. Представляется намного более удобным хранить большие объекты (длинные списки, кортежи и т.д) не как неделимую последовательность байтов, поддерживая деление на записи извне, а интерпретировать большой объект как совокупность скалярных значений переменной длины.

Другой известной объектно-ориентированной системой является GemStone [3]. Каждый объект в системе хранения GemStone разбивается на множество элементов ограниченного размера, для кластеризации которых презназначен менеджер объекта.

Другой схожей системой является система хранения для iMAX- 432 [4]. Недостатком этой системы является то, что она расчитана на объекты до 500 байтов.

Система хранения POSTGRES [5] основана на использовании кортежей и отношений. Каждый кортеж индентифицируется уникальной последовательностью 64 битов, которые никогда не изменяются. Кортежи не меняют своего местоположения. Модифицированный кортеж, с той же последовательностью битов и новой временной отметкой добавляется в базу данных. "Вакуумный процесс"перемещает старые данные на архивный диск. Хоть POSTGRES и поддерживает составные объекты, не обеспечен четкий механизм для хранения и манипулирования большими объектами.

Приложению удобнее интерпретировать объекты как последовательность скалярных значений переменной длины, а не как однородную последовательность байтов. Именно этот подход и реализован в предлагаемом методе хранения: объект рассматривается как последовательность скаляров переменной длины, и модификация объекта производится на уровне скалярных значений. Скалярные значения размещаются в последовательности сегментов равного размера.

3 Структура объекта хранения

Объект хранения представляет собой совокупность упорядоченных скалярных значений переменной длины, размещенных в сегментах данных. Структура данных, используемая для индексирования сегментов, близка к структуре B-дерева, сгенерированного, как и в системе EXODUS, на основе "порций". Отличие заключается в том, что в методе хранения системы EXODUS "порция"представляет собой байты объекта, а в предлагаемом методе - скалярные значения переменной длины. Каждый индексный блок и блок данных занимают по одной дисковой странице. Индексное дерево построено по следующии принципам:

1. Каждый индексный блок имеет ссылку на предыдущий индексный блок.
2. Каждый индексный блок имеет ссылку на левый соседний блок.
3. Каждый индексный блок имеет ссылку на правый соседний блок.
4. Каждый индексный блок должен быть заполнен не менее, чем наполовину, т.е. содержит не менее $((BL-16)/8)/2$ индексных пар, где BL - размер дисковой страницы.
5. Каждая индексная пара содержит количество записей в страницах данных, на которые ссылается эта индексная ветка, и указатель на следующий блок.
6. На каждую запись имеет ссылку только одна индексная ветка (если запись целиком не помещается в одном блоке данных, то индексная ветка ссылается на блок данных, содержащий начало этой записи).

Оглавление индексного блока содержит, на первый взгляд, избыточную информацию, но столь подробное оглавление чрезвычайно удобно как при модификации объекта хранения, так и при доступе к различным скалярным значениям, и позволяет значительно сократить число обращений к диску.

Блок данных содержит:

- набор данных;
- оглавление:
  - количество байтов, составлющее набор данных;
  - указатель на начало первой записи.

Каждое скалярное значение, входящее в набор данных, состоит из длины и непосредственной последовательности байтов.

4 Операции с объектом хранения

4.1 Поиск

Операция поиска позволяет получить доступ к последовательности N записей, следующих за записью с номером S.

1. Положим пару индексного блока (количество записей в следующем блоке; указатель на следующий блок) равной $c[i], p[i]$.
2. Положим $start = S$, $sum = 0$, P - вершина индексного дерева.
3. Пока P - индексный блок, делаем следующее:
   - последовательно перебираем индексные пары блока,
   - подсчитывая $sum = sum + c[i]$, пока $sum < start$;
   - полагаем $start = start - sum$, $P = p[i]$ и считываем следующий блок P.
4. Когда P - блок данных,
   - полагаем $delta =$ начало новой записи;
   - пока $start > 0$ вычисляем $len =$ длина записи,
   - $delta = delta + ln$, $satrt = start - 1$;
5. Когда $start = 0$, проверяем, все ли искомые записи находятся в текущем блоке данных. Если да, то, начиная с $delta$, считываем требуемые записи, сперва определяя по очереди их длину. Если записи целиком не помещаются в текущем блоке данных, начиная с $delta$ считываем весь текущий блок данных, затем следующий блок данных целиком, используя последний считанный индексный блок, и т.д. до тех пор, пока не считаем запись в номером $S + N$.

4.2 Вставка

Операция вставки позволяет добавить последовательность N записей после записьи с номером S. Положим пару индексного блока (количество записей в следующем блоке; указатель на следующий блок) $= c[i], p[i]$.

1. Положим $start = S$, $sum = 0$, P - вершина индексного дерева.
2. Пока P - индексный блок, делаем следующее:
   - последовательно перебираем индексные пары блока, подсчитывая $sum = sum + c[i]$, пока $sum < start$;
   - полагаем $start = start - sum$, $P = p[i]$, увеличиваем $c[i] = c[i] + N$ и считываем следующий блок P.
3. Когда P - блок данных, полагаем $delta =$ начало новой записи; пока $start > 0$ вычисляем $len =$ длина записи, $delta = delta + ln$, $start = start - 1$.
4. Когда $tart = 0$, начиная с $delta$ пробуем добавить N записей в текущий блок данных. Если происходит переполнение блока, то блок данных расщепляется на два блока одинаковой длины, и новый блок добавляется в индексное дерево. При этом блок данных и индексный блок могут быть заполнен не менее, чем на $1/2$.

4.3 Удаление

Операция удаления позволяет удалить последовательность N записей, начиная с записи с номером S. Положим пару индексного блока (количество записей в следующем блоке; указатель на следующий блок) $= c[i], p[i]$.

1. Положим $start = S$, $sum = 0$, P - вершина индексного дерева.
2. Пока P - индексный блок, делаем следующее:
   - полагаем $N_{tmp} = N$;
   - последовательно перебираем индексные пары блока, подсчитывая $sum = sum + c[i]$, пока $sum < start$;
   - полагаем $start = start - sum$, $P = p[i]$, $tmp = maximum(c[i] - N, 0)$, $c[i] = tmp$, $N_{tmp} = N_{tmp} - tmp$;
3. Пока $N_{tmp} > 0$, изменяем количество скольярных значений следующей индексной пары (или следующая индексная пара текущего индексного блока, или первая индексная пара правого соседнего блока). $tmp = maximum(c[i] - N, 0)$, $c[i] = tmp$, $N_{tmp} = N_{tmp} - tmp$. Если при этом $c[i] = 0$, то индексная пара удаляется из индексного блока. При этом если индексный блок оказался заполненным менее, чем на $1/2$, то пытаемся объединить его с соседними индексными блоками. Считываем следуюий блок P.
4. Когда P - блок данных,
   - полагаем $delta =$ начало новой записи;
   - пока $start > 0$ вычисляем $len =$ длина записи, $delta = delta + ln$, $satrt = start - 1$;
   - Когда $start = 0$, полагаем $sum = 0$.
5. Пока $N > 0$, начиная с $delta$ определяем $len =$ длина записи. Если запись целиком попала в блок данных, $sum = sum + len$, $delta = delta + len$, $N = N - 1$. Если все удаляемые записи целиком оказались в одном блоке данных, то модифицируем данный блок, изменяя его оглавление. При этом, если оказалось, что блок данных заполнен менее, чем наполовину, пробуем слить его с соседними блоками. Если же оказалось, что удаляемые записи целиком не попали в один блок данных, модифицируем текущий блок данных, добавляем в список пустых блоков ссылки на блоки данных, которые содержат только удаляемые записи, находим конец последней удаляемой записи, считываем блок данных и модифицируем его.

5 Система хранения

Предлагаемый метод хранения реализован в рамках системы хранения [8]. Для манипулирования объектами хранения система хранения заводит на каждый объект базы данных головной блок. Головной блок включает в себя:

1. Общее количество записей в файле.
2. Общее количество байтов данных в файле.
3. Количество уровней индексного дерева.
4. Указатель на первый индексный блок.
5. Указатель на первый свободный блок.
6. Количество занятых блоков в файле.

Файловая система обеспечивает создание нового объекта хранения, поддерживает индексную систему, хранит предыдущие версии хранимого объекта.

6 Оценки метода

Если сравнивать предлагаемый метод хранения с методом хранения системы EXODUS, то оценки числа обращений к диску при операциях чтения, вставки и удаления будут очень близкими, так как структура индексного оглавления аналогична (отличие заключается в том, что индексное дерево в предлагаемом методе строится не по номерам байтов, а по номерам скалярных значений). Существенное отличие заключается в ином представлении блоков данных. Причина этого в функциональном отличии двух методов: EXODUS представляет данные как неделимую последовательность байтов, а предлагаемый метод рассматривает их как последовательность скалярных значений переменной длины. Коэффициент использования памяти в обоих методах приближенно равен 0.67.

Если же сравнить предлагаемый метод с системой хранения Starburst, то оценки чтения предлагаемого метода будут несколько хуже. В системе Starburst очень высок коэффициент заполнения памяти: для больших объектов он приближается к 100обращений при операциях считывания. Но не следует забывать, что система Starburst не рассчитана на поддержку вставки и удаления из середины объекта. При вставке или удалении из середины оставшаяся часть большого объекта переписывается. Кроме того, эта система, так же, как и EXODUS, представляет большой объект как однородную последовательность байтов.

В системе EOS операции считывания и модификации, возможно, будут выполняться эффективнее, так как коэффициент использования памяти в этой системе несколько выше, чем у предлагаемого метода: приблизительно 0.8. Но и в этой системе нет нужной функциональности: сложно работать с большими объектами, состоящими из скалярных значений переменной длины.

Допустим следующие обозначения:

BL - размер дисковой страницы;

s - число дисковых страниц в одном сегменте;

$s * BL$ - размер сегмента данных;

n - число дисковых страниц в файле;

B - число байтов в файле;

m - число записей в файле;

B/m - средняя длина записи;

$2d + 1$ - максимальное число индексов в индексном сегменте;

$2e + 1$ - максимальное число записей в сегменте данных;

$coef$ - коэффициент использования памяти;

$$e = \frac{coef * s * BL}{2 * B/m}$$

$$d = \frac{coef * BL}{2 * B/m}$$

При таких обозначениях оценка чтения для ESM и EOS будет $\log_d n/e$, а для Starburst - $\frac{B/m}{BL}$. Про оценки обновления можно сказать следующее: для EOS и ESM в среднем оценка будет $2 * \log_d n/e$, а в самом худьшем случае - $3 * \log_d n/e$. Оценка обновления же в системе Starburst пропорциональна длине объекта: $\frac{B}{2*BL}$.

Список литературы

[1] M.G. Carey, D.J. DeWitt, J.E. Richardson, E.J. Shekita. "Object and File Management in the EXODUS Extensible Database System", Proceedings of the Twelfth International Conference on Very Large Data Bases.

[2] O. Deux et al. "The O2 SYSTEM", Communication of the ACM, October 1991, Vol.34, No.10, P.34-49.

[3] Capeland, G. and D. Maier. "Making Smalltalk a Database System", Proceedings of the 1984 SIGMOD Conference, Boston, MA, May 1984.

[4] D. Reed. "Implementing Atomic Actions on Decentralized Data", ACM Transactions on Computer System 1, 1, March 1983.

[5] M. Stonebraker, G. Kemnitz. "The POSTGRES Next-Generation Database Management System", Communication of the ACM, October 1991, Vol.34, No.10, P.34-49.

[6] G.M. Lohman, B. Lindsay, H. Pirahesh, K.B. Schiefer. "Extensions to STARBURST: Objects, Types, Functins, and Ruls", Comunications of the ACM, V.34, N.10, October 1991, pp.94-109.

[7] A. Biliris. "An Efficient Database Storage Structure for Large Dynamic Objects", Proc. IEEE 8th International Conference on Data Engineering, Phoenix, Arizona, February 1992, pp.301-308.

[8] Б.А. Новиков. "Реализация сложных объектов в системе хранения", Управляющие Системы и Машины, 7, 1991, С.46-52.