Обработка и оптимизация запросов в базах данных

Специальный курс для студентов 4-5 курсов (отделение информатики)

Б.А. Новиков

Осенний семестр 2006 года

http://meta.math.spbu.ru/~boris/courses/

Об этом курсе

Техника безопасности при использовании проекционного монитора

Соотношение спецкурса и спецсеминара

Подготовка докладов

Соотношение спецкурса и спецсеминара

Спецкурс включает базовые сведения по обработке и оптимизации запросов, в основном это стабильный материал.

Спецсеминар дополняет материал спецкурса по отдельным темам, связанным с актуальными направлениями исследований.

Программа семинара обновляется ежегодно.

Для успешной сдачи экзамена или зачета по спецкурсу необходимо активное участие в семинаре.

Для получения зачета по спецсеминару необходимо и достаточно сделать доклад.

Подготовка слайдов для доклада на семинаре

Каждый доклад готовится на основании статьи.

Слайды должны быть читаемыми, просто показывать фрагменты статьи недостаточно.

Можно исползовать MS PowerPoint или LaTeX (PDF)

`http://meta.math.spbu.ru/~boris/LaTeXslides/`

Содержание курса

Введение: архитектура средств обработки запросов в СУБД

Структуры хранения данных и индексов

Алгоритмы манипулирования данными

Оптимизация запросов

Параллельные базы данных

Специальные типы данных

Литература по курсу

Ullman H. Principles of database and knowledge basesystems. Computer Science Press, 1988.

Graefe. G. Query evaluation techniques for large databases. ACM Comp. Surveys 26:2, 1993.

Ioannidis, Y.: Query Optimization

`http://infolab.stanford.edu/~widom/cs346/ioannidis.pdf`

Б.А. Новиков, Г.Р. Домбровская Настройка приложений баз данных. БХВ: СПб. 2006.

Much more . . .

Роль баз данных

Значение постоянно хранимых данных для приложений определяется высокой стоимостью данных

Выполнение запросов — основная функция систем управления базами данных

Качество и стоимость СУБД определяются ее способностью выполнять запросы на больших объемах данных и зависят главным образом от характеристик обработчика запросов

Место СУБД в архитектуре современных распределенных систем

В традиционных двухуровневых архитектурах СУБД выполняют разнообразные функции: идентификация пользователей, безопасность данных, разграничение доступа, независимость данных и др.

В многоуровневых архитектурах многие функции перенесены в сервер приложений

Роль СУБД сводится к поддержке согласованности и выполнению запросов

Архитектура средств обработки запросов

Выполнение запросов - основная функция СУБД как серверов данных

Клиенты используют высокоуровневый язык запросов

Соотношение исчисления и алгебры (например, реляционной)

Компилятор запросов

Оптимизация: выбор плана выполнения запроса

Обработчик запросов: интерпретатор алгебры

Структуры хранения: данные и индексы

Производительность

Пропускная способность: количество запросов, выполняемых за единицу времени

Время ответа: время выполнения отдельного запроса (среднее, максимальное)

Количество обращений к внешним запоминающим устройствам

Время процессора в этом курсе чаще всего несущественно

2 Структуры хранения и индексы

Структуры хранения в СУБД

Одномерные индексы: деревья и хеширование

Многомерные индексы

Использование оперативной памяти

Структуры хранения

Логическая и физическая организация данных

Аппаратные ограничения

Методы хранения коллекций

Адресация данных

Логическое и физическое

Структура данных, видимая приложениям, отображается на структуру хранения (коллекции и объекты).

Коллекции и объекты размещаются в экстентах, состоящих из блоков.

Выделение места для хранения коллекций может выполняться СУБД (Oracle) или файловой системой (IBM DB2).

Параллельные СУБД используют более сложные схемы размещения данных.

Аппаратные ограничения

Основной вид носителя - диски

Форматирование дисков и логические блоки в СУБД: размер блока кратен размеру сектора.

Реальная структура диска обычно скрыта от СУБД

Характеристики производительности: (среднее) время произвольного доступа и время последовательной обработки смежных участков

Размещение коллекций

Плотное размещение: объекты размещаются без упорядочивания, заполняя выделенные блоки в порядке добавления.

Совместное размещение: взаимосвязанные объекты разных коллекций размещаются вместе (в одном или близких блоках)

Упорядоченное размещение: объекты размещаются в соответствии с упорядочением по некоторому ключу.

Разделяемые кластеры: несколько коллекций размещаются на основе общего ключа

Хеш-кластеры.

Размещение объектов в блоках

Независимо от способа распределения памяти, СУБД рассматривает память как набор блоков фиксированного размера (2–64 K).

Проблема размещения объектов (переменной длины) в блоках:

Перемещаемые или неперемещаемые объекты

Изменение длины объектов

Управление свободной памятью в блоке

Типичный метод организации блока: объекты размещаются начиная от начала блока, а таблица смещений – с конца.

Размещение в блоках - 2

Обычное размещение "объект за объектом"неэффективно, если выбираются только значения отдельных атрибутов (плохо используется кэш процессора).

Альтернатива: значения атрибутов из разных объектов, размещенных в блоке, хранятся компактно (в мини-блоке переменного размера).

Размещение коллекций: master-detail

Две взаимосвязанные коллекции: небольшая master, большая detail.

Раздельное размещение: быстрая обработка master

Совместное размещение: медленная master, быстрая совместная обработка

Раздельные кластеры: быстрая частичная совместная обработка

Совместный кластер: быстрая выборка по диапазону ключей

Адресация

Абсолютная или относительная адресация:

на диске (MMBBCCHHR, TTR или номер сектора)

В файловой системе: Номер байта от начала файла

В пространстве коллекции: (номер блока, смещение)

Символические и позиционные указатели

Перемещаемые или фиксированные объекты?

Индексы

Индексом называется избыточная структура данных, предназначенная для ускорения поиска.

Объект индекса: (ключ, данные). Обычно “данные” — список ссылок.

Структура ключа: одномерные и многомерные индексы

Упорядоченные или неупорядоченные

Типы запросов: точечные, по диапазону, приблизительные, по подбию.

Характеристики: время доступа/обновления, необходимость реорганизаций, коэффициент использования памяти.

Одномерные индексы

Индексы с упорядочением: деревья

Быстрые индексы без упорядочивания: хеширование

Битовые индексы

Гибридные методы одномерного индексирования

Одномерные упорядоченные индексы

Двоичные деревья (во всех модификациях) непригодны для внешней памяти

Деревья, учитывающие структуру диска:

Выделенная память состоит из цилиндров, в каждом цилиндре одна (или несколько) индексных дорожек, в остальных размещаются записи индекса.

(возможно, многоуровневый) индекс цилиндров.

Очень быстрый поиск, зависимость от физических характеристик устройства, необходимость реоранизации.

B-деревья

Сбалансированная структура, не зависящая от характеристик устройств.

При ветвлении m время поиска составляет $\log_m(N/m)$ (N - число индексных записей).

Процедура построения добавлением отдельных записей

Не требуются реорганизации

Использование памяти: в каждом блоке $0.5 < s \leq 1$, в среднем $\ln 2 \approx 0.69$.

Варианты и улучшения

B^+-деревья: записи данных только в листьях.

Преимущества: предсказуемое ветвление, меньшее количество уровней.

Механизм переливания для улучшения использования памяти.

Управление свободными блоками

Сжатие ключей слева и справа

авиация	0 авиация	0 3 ави
автомат	2 томат	2 5 томат
автоматический	7 ический	7 1 и
автоматный	7 ный	7 1 н
автомобиль	5 обиль	5 5 обиль
автомобильный	10 ный	10 1 н
агрегат	1 грегат	1 1 г
блок	0 блок	0 1 б

Одномерные неупорядоченные индексы: хеширование

Функция хеширования $H(key) \rightarrow Location$ преобразует пространство ключей в пространство адресов размещения индексных записей.

Составляющие метода хеширования:

выбор функции хеширования

метод обработки коллизий

структура пространства, в котором размещяются записи.

Функции хеширования

Требуется высокая равномерность распределения ключей по пространству.

Затраты процессорного времени менее существенны.

Деление с остатком

Вычисление полинома

Суммирование с частичным инвертированием последовательности битов

Десятки других методов опубликованы в литературе.

Обработка коллизий

Цепочки переполнения в отдельной области памяти

Цепочки переполнения в основной области памяти

Открытая адресация: записи переполнения размещаются в основной области памяти на ближайшее свободное место. Поиск последовательным просмотром.

Структура пространства индексных записей

Адресация с точностью до записи приводит к большому количеству коллизий

Адресация с точностью до блока: коллизии возникают только при переполнении блока.

Характеристики методов хеширования с фиксированным пространством

При отсутствии коллизий (совершенное хеширование) время доступа равно 1.

Количество коллизий зависит от заполненности файла.

Проблемы:

(1) Низкая заполненность на начальной фазе

(2) Деградация производительности при высокой заполненности

(3) Необходимы периодические реорганизации.

Расширяемое хеширование

Количество используемых битов функции хеширования определяется текущим уровнем файла l, начальное состояние $l - 0$.

Для адресации используется вспомогательная таблица, размещаемая в оперативной памяти.

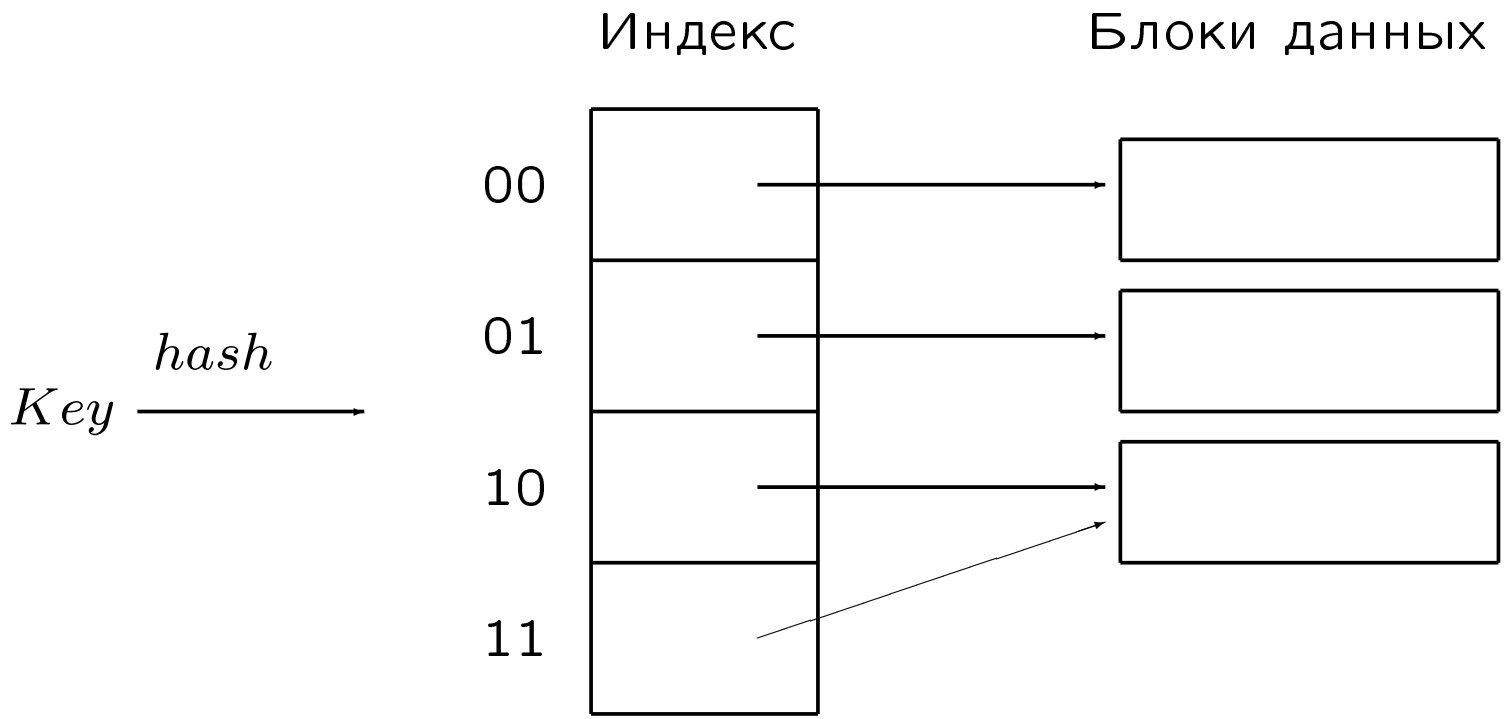
Биты	Адрес	l
0	A1	0

Биты	Адрес	l
0	A1	1
1	A2	1

Биты	Адрес	l
00	A1	2
01	A3	2
10	A2	1
11	A2	1

При расщеплении блока максимального уровня размер таблицы удваивается.

Расширяемое хеширование



Расширяемое хеширование - расщепление

Биты	Адрес	l
000	A1	2
001	A1	2
010	A3	3
011	A4	3
100	A2	1
101	A2	1
110	A2	1
111	A2	1

Биты	Адрес	l
000	A1	2
001	A1	2
010	A3	3
011	A4	3
100	A2	2
101	A2	2
110	A5	2
111	A5	2

Расширяемое хеширование - характеристики

Время доступа: всегда 1 (совершенное хеширование)

Использование памяти: $\ln 2$

Недостатки:

Низкое использование памяти

Быстрый рост таблицы в оперативной памяти при неравномерностях распределения значений функции хеширования.

Расширяемое хеширование с ограниченным уровнем

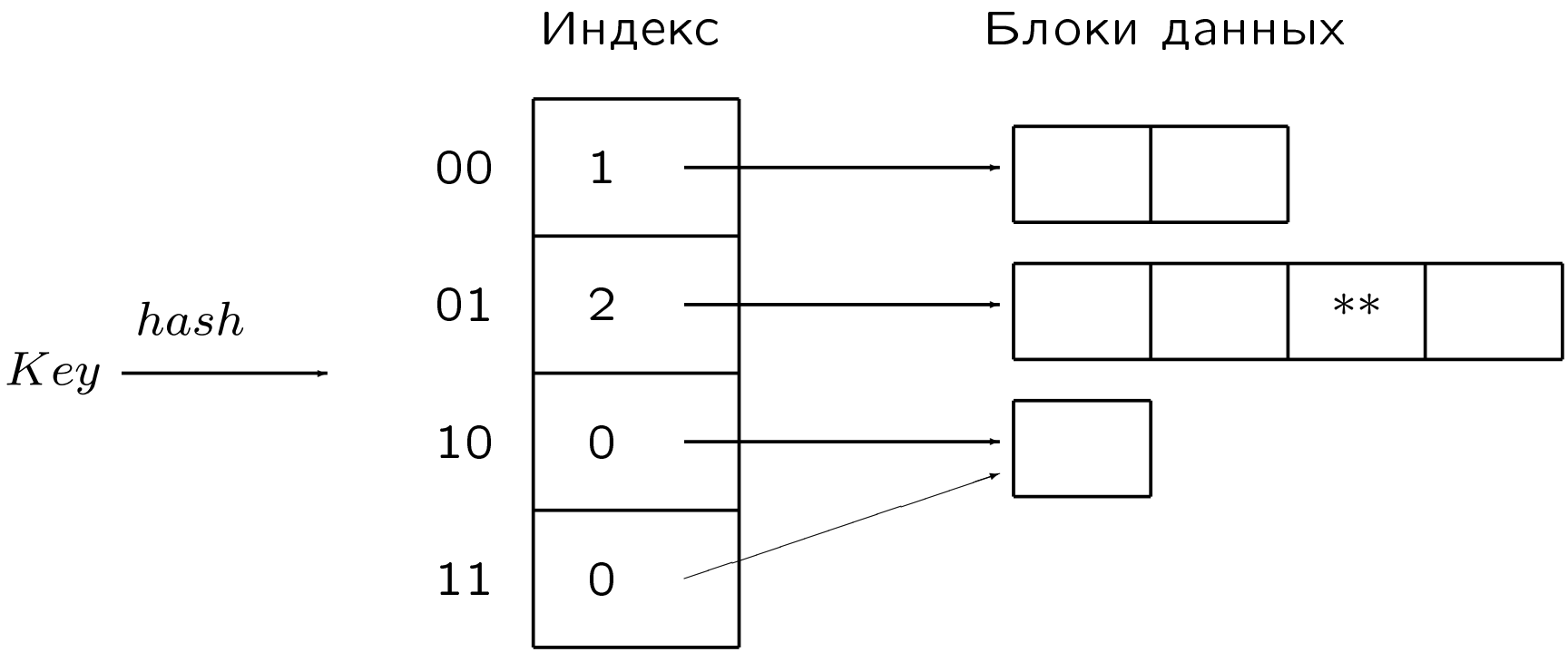
При создании файла определяется предельный уровень L, до которого может вырасти таблица.

При необходимости дальнейшего расщепления блоков уровня L эти блоки заменяются на экстенты, состоящие из $2, 4, \ldots 2^s$ блоков.

Для адресации в экстентах используются следующие s, битов функции хеширования.

Усложненное управление распределением блоков, невысокий коэффициент заполнения.

Расширяемое хеширование с ограниченным индексом



Расширяемое хеширование с деревом таблиц

Рост таблицы ограничивается заданным уровнем L.

При необходимости расщепления последнего уровня ссылка на блок заменяется ссылкой на новую таблицу, использующую биты $L+1, \ldots 2L$.

Количество уровней таблиц не ограничено.

Размер каждой таблицы может быть небольшим (например, $L = 5$).

Ожидается хорошее поведение на неравномерно распределенных данных.

Линейное хеширование

Последовательность функций $h_i(k), i = 0, 1, \ldots$

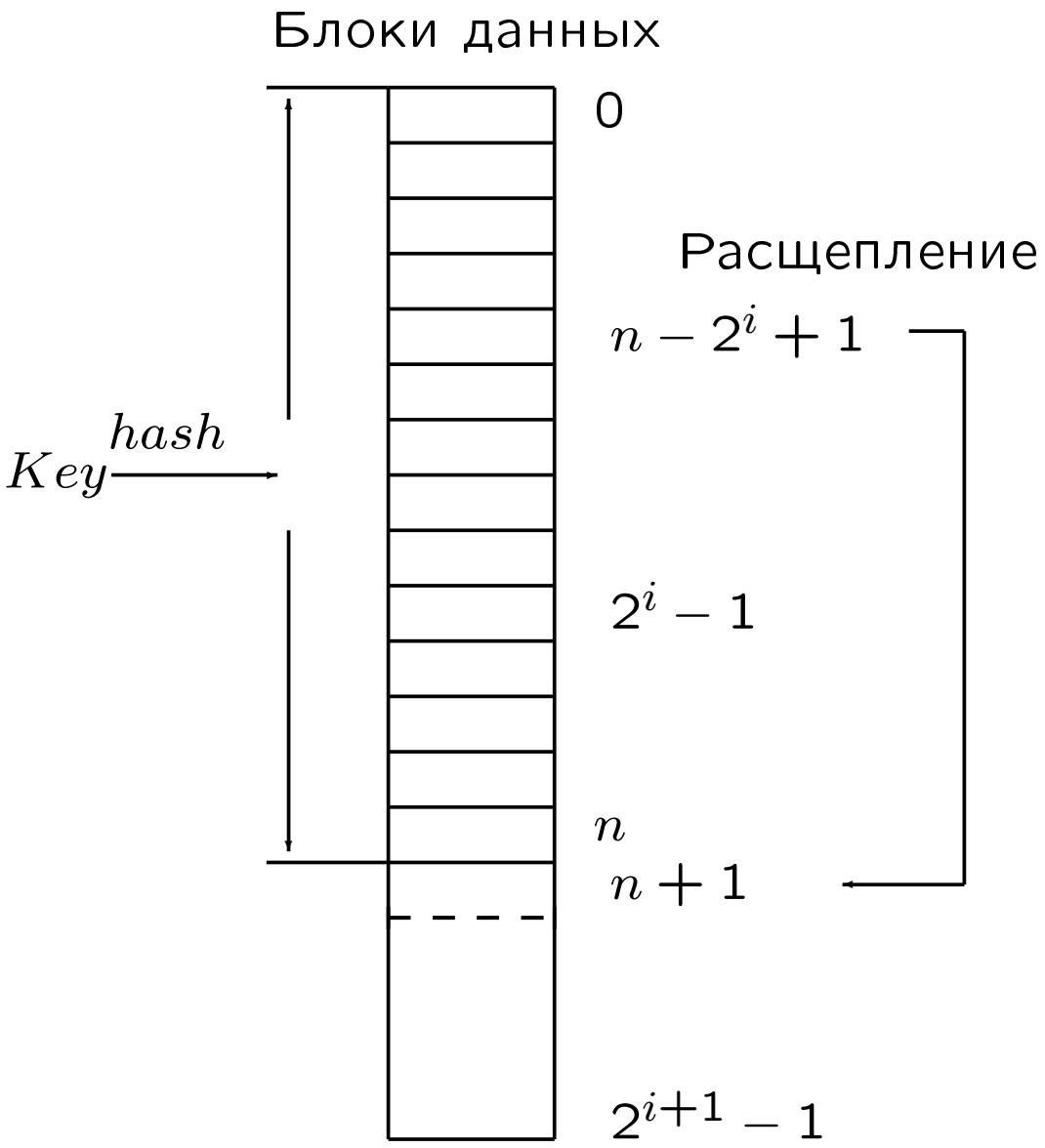
$h_0(k) = 0, \quad 0 \leq h_i(k) < 2^i$

$h_{i+1} = \begin{cases} h_i(k) \\ h_i(k) + 2^i \end{cases}$ с равной вероятностью

Пусть в текущем состоянии файл содержит B блоков, $2^l \leq B < 2^{l+1}$.

Функция хеширования $H(k) = \begin{cases} h_{l+1}(k), & h_{l+1}(k) \leq B \\ h_l(k) & \text{иначе.} \end{cases}$

Линейное хеширование



Линейное хеширование - рост файла

В начальном состоянии $l = 0, B = 1$.

При необходимости расщепления расщепляется блок с номером $B + 1 - 2^l$ в соответствии с функцией хеширования для $B + 1$.

Данные попадают в блоки $B + 1 - 2^l$ и $B + 1$.

Если при этом $B + 1 = 2^{l+1}$, то l увеличивается на 1.

Функция хеширования $H(k) = \begin{cases} h_{l+1}(k), & h_{l+1}(k) \leq B \\ h_l(k) & \text{иначе.} \end{cases}$

Линейное хеширование: уточнения и варианты

Возможны коллизии. Цепочки переполнения можно размещать в основной области с конца, где блоки менее заполнены.

Нужно учитывать цепочки при расщеплении.

Можно использовать шаг, отличный от 2^i.

При больших размерах файла целесообразно при каждом расщеплении расщеплять несколько блоков.

Линейное хеширование: характеристики

Управляемый коэффициент заполнения

Баланс между заполнением и временем доступа.

Минимальное совершенное хеширование

Функция хеширования называется совершенной на множестве ключей K, если все ее значения различны, и минимальной, если размер адресного пространства равен мощности K.

Для любого множества ключей существуют минимальные совершенные функции хеширования.

$$h(x) = \sum_i i \frac{x}{k_i} \prod_{j \neq i} \frac{(x - k_j)}{(k_i - k_j)}$$

Известны намного более эффективные методы построения минимальных совершенных функций хеширования.

Применение: для тиражирумых индексов ROM.

Составное совершенное хеширование

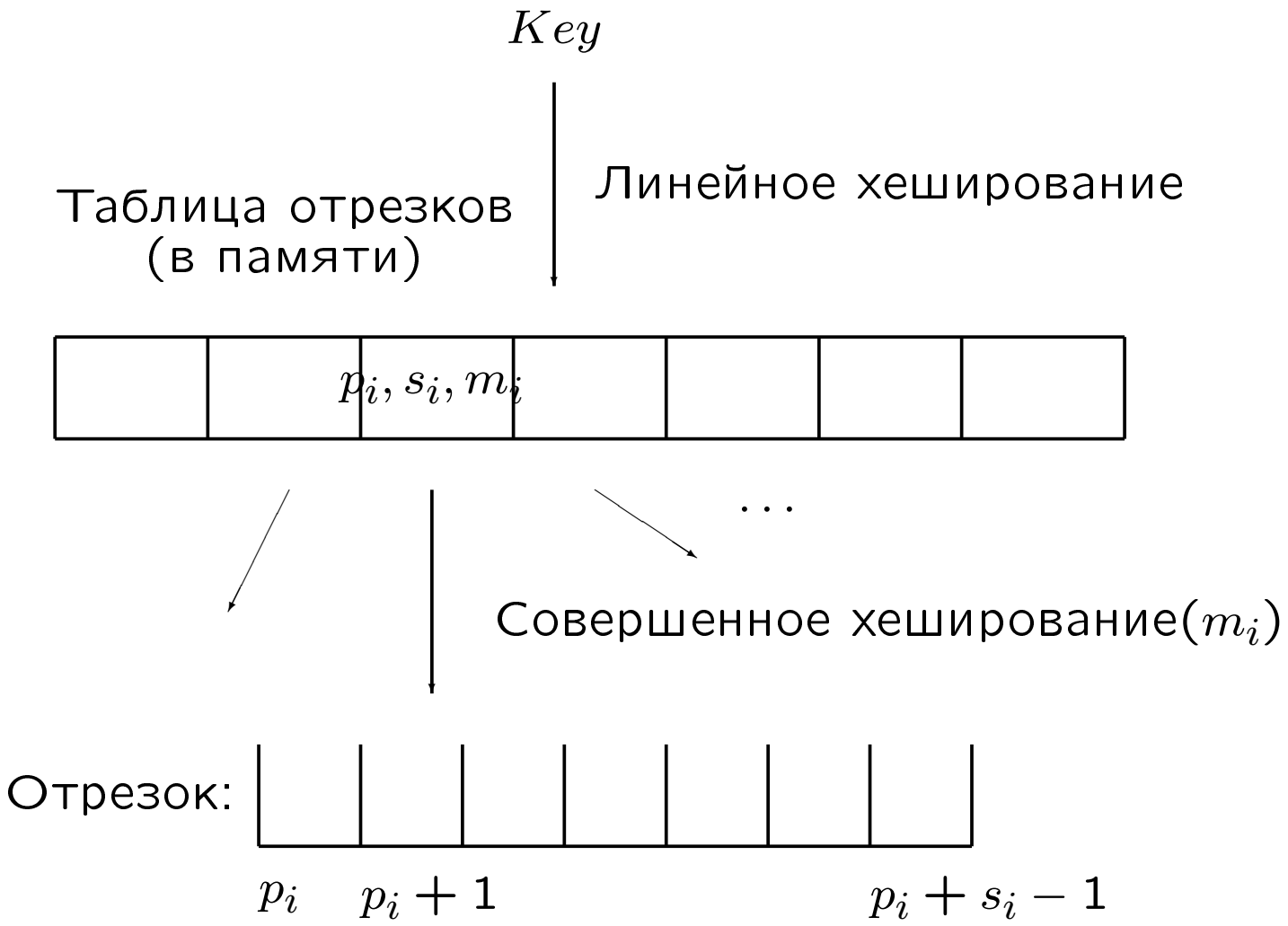
Двухуровневая структура данных.

На первом уровне используется динамическое хеширование (например, линейное хеширование), результат которого - позиция в таблице, хранящейся в оперативной памяти.

Строка этой таблицы: (A_i, B_i, h_i) (адрес экстента, размер экстента, функция хеширования для этого экстента).

На втором уровне применяется своя функция хеширования для каждого экстента.

Составное совершенное хеширование



Составное совершенное хеширование - 2

Выбор функций хеширования второго уровня: параметры функции должны занимать мало места, например, деление с остатком.

Процедура вставки: Если на втором уровне нет коллизии, то выполняется вставка, в случае коллизий делается попытка подобрать другую функцию хеширования.

В случае неудачи - увеличение экстента и подбор новой функции.

Очень большие экстенты неэффективны, поэтому нужно использовать расщепление на первом уровне.

Составное совершенное хеширование: варианты

B_+-дерево вместо первого уровня. Частичное сохранение порядка ключей, возможен поиск по диапазону.

Индекс для экстентов второго уровня: в таблице хранится минимальное значение функции хеширования для каждого блока экстента. Упрощается подбор функции хеширования за счет большего расхода памяти.

Интерполяционное индексирование

Линейная функция хеширования: $\frac{x-K_{min}}{K_{max}-Kmin}$ сохраняет порядок, но не обеспечивает равномерность распределения значений.

Можно выбрать последовательность ключей

$$k_0 = k_{min}, k_1, k_2, \ldots k_l = k_{max}$$

и использовать линейную интерполяцию на каждом интервале $[k_i, k_{i+1}]$.

$$k_i <= k < k_{i+1} \quad h(k) = A_i + (A_{i+1} - A_i)\frac{k - k_i}{k_{i+1} - k_i}$$

Интерполяционное индексирование - 2

Интервалы выбираются таким образом, чтобы в каждом было примерно одинаковое количество фактически имеющихся ключей.

Для каждого интервала ключей выделяется свой диапазон адресов блоков.

Индекс сохраняет порядок ключей.

Интерполяционное индексирование - 3

Для разрешения коллизий последние записи блока перемещаются в следующий блок.

Хранение последовательности разделяющих ключей: индекс как в B_+-дереве.

Преимущество: меньшая высота дерева.

Индексы на основе битовых шкал

Рассмотренные типы индексов плохо работают при высокой селективности запросов и/или при малом количестве различных значений ключа.

Битовый индекс: для каждого значения ключа строится битовая шкала с 1 в позициях, соответствующих записям, содержащим это значение.

Битовые индексы высокоэффективны для сложных критериев, использующих значения многих атрибутов, каждый из которых имеет мало значений.

Многомерные методы доступа

Составные ключи

Использование одномерных методов для одномерных объектов

Методы для точечных объектов

Методы для протяженных объектов

Использование одномерных методов

Упорядоченные индексы по составному ключу: отсутствует симметрия размерностей.

Хеширование по составному ключу: для каждой составляющей ключа выделяется несколько битов в значении, вырабатываемом функцией хеширования.

При поиске по неполному ключу необходимо просматривать один или несколько диапазонов индекса.

Инвертированные файлы

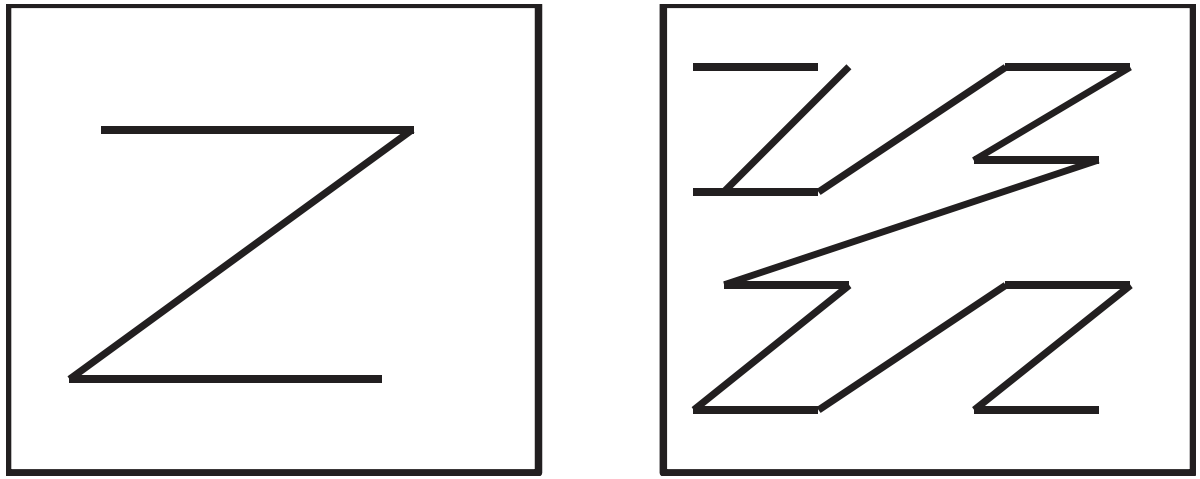
Набор обычных упорядоченных индексов для каждого ключа отдельно.

При поиске по нескольким ключам выполняются теоретико-множественные операции над списками ссылок на объекты данных.

Если проиндексированы все атрибуты, нет необходимости хранить объекты данных.

Возможно сочетание обычных индексов (B_+-деревьев) с индексами на основе битовых шкал.

Z-упорядочение



Никакое отображение многомерного пространства в пространство другой размерности не может сохранить отношение близости.

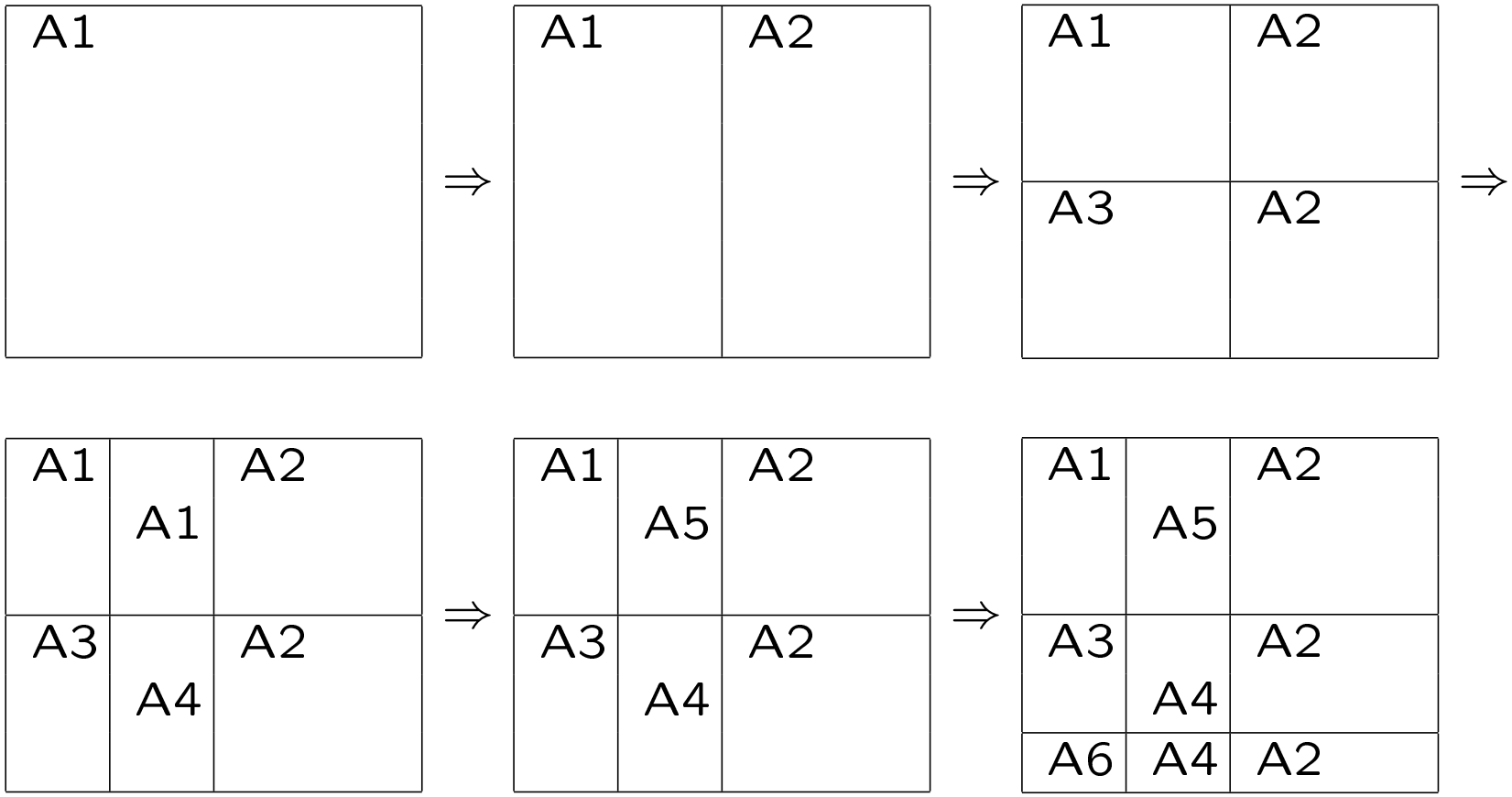
Индексирование точечных объектов: Grid-файл

Цели: симметричная структура данных с быстрым доступом при малом объеме результата.

Любой точечный запрос (заданы все атрибуты составного ключа) выполняется за 2 обращения к диску (оглавление и данные).

Предполагается, что каждое измерение имеет достаточно много различных значений.

Клеточные файлы - расщепление



Клеточный файл: структура оглавления

Разделяющие значения ключей (шкалы по каждой размерности) могут храниться в оперативной памяти.

Оглавление может быть представлено многомерным массивом.

Допускаются только выпуклые зоны (группы клеток, ссылающиеся на один блок).

Выбор размерности для очередного сечения.

Характеристики клеточных файлов

Разбиение пространства, а не имеющихся данных.

Возможен поиск по произвольной комбинации диапазонов и точечных значений по различным ключам.

Точечный поиск всегда выполняется за 2 доступа к диску.

Недостатки клеточных файлов

Быстрый рост оглавления при неравномерном распределении данных.

Быстрый рост оглавления при увеличении размерности.

Вариант: Иерархическая структура вложенных клеток (BANG-файл).

Индексирование протяженных объектов

Для включения в индекс объекты заменяются более простыми.

Параллелепипед с ребрами, параллельными осям координат.

Достаточно хранить координаты максимальной диагонали.

Сведение к точечным объектам: пара точек (концы диагонали) представляется точкой в пространстве вдвое большей размерности.

R-деревья

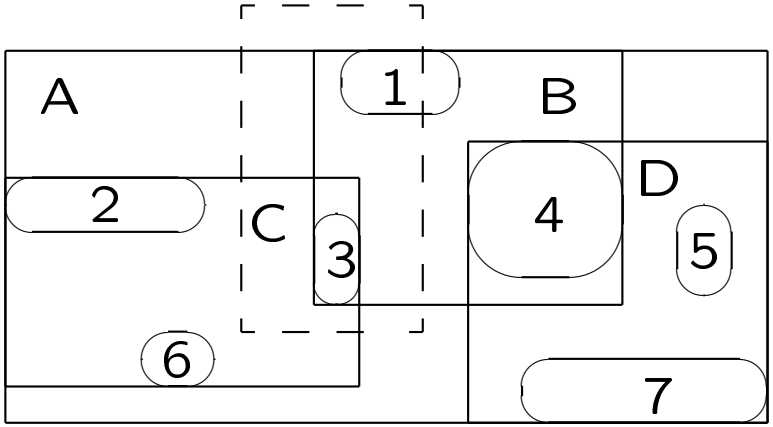
Многомерный аналог B-деревьев.

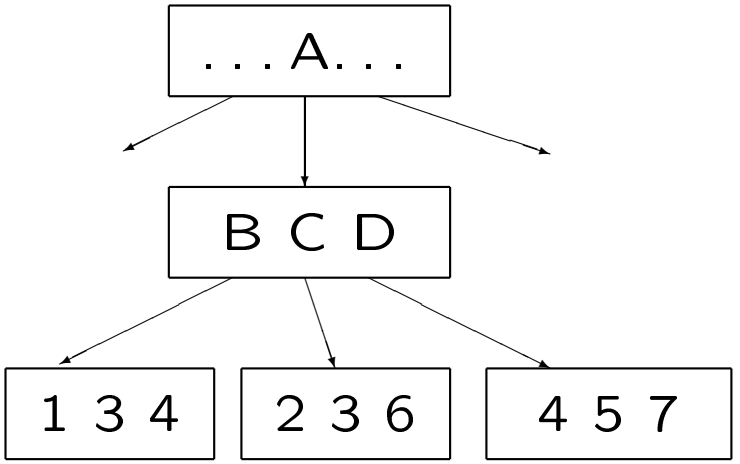
Протяженные объекты аппроксимируются параллелепипедами с ребрами, параллельными осям координат.

Несколько близких объектов размещаются в одном блоке, который представляется охватывающим параллелепипедом для включения в блок более высокого уровня.

Допускается пересечение охватывающих параллелепипедов.

R-дерево: пример





R-дерево: расщепление

Выбор направления сечения параллелепипеда.

Распределение объектов между новыми узлами неоднозначно.

Возможные критерии:

Минимальный объем

Минимальный периметр

Максимум расстояния между центрами

R-деревья: варианты и дополнения

R_+-деревья: охватывающие прямоугольники не пересекаются. Объекты включаются в каждый из прямоугольников, с которым они пересекаются.

При поиске более точный спуск по каждой ветви, но требуется просмотр большего количества ветвей.

Повторная вставка: объекты, расположенные на границе, удаляются из дерева и вставляются снова, возможно, в другой блок.

R-деревья: проблемы

Объекты неправильной формы или большой протяженности, но малой площади плохо аппроксимируются прямоугольниками.

При увеличении размерности эффективность очень быстро падает, а вычислительная сложность возрастает.

C-деревья

Обобщение R_+-деревьев: вместо прямоугольников используются выпуклые многоугольники (с произвольным направлением сторон).

Объект представляется конечным набором покрывающих непересекающихся выпуклых многоугольников.

Расщепление: выбирается гиперплоскость, пересекающая многогранник, соответствующий вершине.

При расщеплении некоторые многогранники объектов могут быть разрезаны на два.

C-деревья - 2

Значительно более точная аппроксимация объектов сложной формы, чем в случае R-деревьев.

Очень велика вычислительная сложность всех операций.

Другие проблемы, связанные с индексированием

Индексирование больших объектов

Индексы для сложных объектов

Индексирование данных очень высокой размерности

Индексирование текстов

3 Алгоритмы выполнения операций алгебры

Выполнение операций селекции

Произведение и естественное соединение

Группировки и проекции

Вычисление транзитивного замыкания

Компиляция запросов

Запрос на высокоуровневом языке транслируется в план выполнения запроса, представляющий собой дерево алгебраического выражения.

Оптимизатор выбирает один из возможных планов.

Для каждой операции выбирается алгоритм ее выполнения.

Исполнитель запросов

Каждая операция интерпретируется исполнителем запросов.

Промежуточные результаты между операциями не записываются, а поступают на вход следующих операций (pipes).

Некоторые операции записывают промежуточные результаты во временные коллекции (например, сортировка).

Операции селекции

Поиск по индексу с последующим доступом по указателям. Сложность: доступ к индексу + число найденных объектов.

Полный просмотр. Сложность: число блоков, занятых коллекцией.

Поиск по индексам эффективен при малой селективности.

Операции сортировки

Операции внешней сортировки должны рассматриваться в других курсах.

В качестве внешней сортировки обычно используется многопотоковая сортировка слиянием.

Прямое произведение

Произведение коллекций вычисляется алгоритмом "вложенные циклы"(NL).

Варианты: циклы по обектам или по блокам.

Сложность определяется размером результата.

Доминирующая компонента — произведение размеров исходных коллекций (в блоках).

Операция естественного соединения: NL

Алгоритм вложенных циклов с одновременной селекцией.

Сложность пропорциональна произведению размеров исходных коллекций (в общем случае).

Соединение: NL для кластеризованных коллекций

Частный случай: естественное соединение по ключу кластера.

Вложенный цикл необходим только в пределах каждого кластера.

Сложность пропорциональна размеру кластера.

Соединение слиянием

Первая фаза: каждая из входных коллекций сортируется по атрибуту соединения.

Вторая фаза: алгоритм слияния упорядоченных коллекций, вложенный цикл для каждого значения ключа.

Сложность: сортировка N log N с дополнительным просмотром.

Частный случай: одна из коллекций кластеризована до начала операции

Соединение: Hash Join

Первая фаза (распределение): коллекции хешируются по ключу соединения и размещаются во временной области памяти (строится хеш-кластер соединяемых коллекций).

Вторая фаза: выполняется соединение по отдельности для каждого значения ключа.

Сложность: линейно зависит от размеров исходных коллекций.

Проекция и агрегирование

При выполнении проекции необходимо устранение дубликатов.

Как устранение дубликатов, так и агрегирование можно рассматривать как особый случай операции соединения.

Алгоритмы: NL, Merge, Hash.

Сложность: как для операции соединения.

4 Оптимизация запросов

Архитектура оптимизатора запросов

Оптимизация селекций

Оптимизация соединений

Пример базы данных

```
emp(name, age, sal, dno)
dept (dno, dname, floor, budget, mgr, ano)
acct(ano, type, balance, bno)
```

Запрос:

```
Select name, floor
   from emp e, dept d
   where e.dno = d.dno and sal > 100K
```

Пример: характеристики схемы хранения

Таблица	строк	блоков
emp	100000	20000
dept 100	10	
sql>100K	10	

Индексы: emp(sal) - B_+-Дерево, dept(dno) - hash-кластер.

Пример: варианты плана

1. Выбрать из emp, используя индекс, NL-join используя индекс dept(dno).

2. Цикл по блокам dept: для каждого блока прочитать emp с последующей фильтрацией по sal>100k.

3. Вложенный цикл по строкам с последующей фильтрацией.

Количество операций чтения диска: 23, 200 000, 10 000 000.

Архитектура оптимизатора

Rewriter: преобразования на уровне исчисления

Генератор алгебраического пространства

Генератор пространства методов и структуры

Модель стоимости

Оценки распределения значений

Planner

Ограничение размеров алгебраического пространства

SPJ-запросы.

Селекции и проекции всегда выполняются при первичном сканировании хранимых таблиц.

Проекции совмещаются с выводом результата других операций.

После применения этих правил достаточно рассматривать только операции соединения.

Ограничение алгебраического пространства 2

Прямое произведение вычисляется только, если оно должно присутствовать в окончательном результате.

Один из операндов операции соединения должен быть хранимой таблицей, а не промежуточным результатом.

Planner: динамическое программирование 1

Частичные планы строятся итеративно (по количеству соединяемых отношений).

Для алгоритма merge-join важно упорядочение таблиц на входе и он вырабатывает упорядоченный результат, поэтому каждый частичный план дополняется информацией об упорядоченности результата.

Частичные планы, вырабатывающие одинаковый порядок, называются эквивалентными.

Динамическое программирование 2

Для каждого отношения перечисляются все возможные методы сканирования (совмещенные с селекциями и проекциями).

Для каждого метода выписывается вырабатываемое упорядочение. Вычисляется оценка стоимости для каждого полученного частичного плана.

Для каждого класса эквивалентности выбирается лучший план. Неупорядоченный частичный план сохраняется, если он лучше любого из упорядоченных.

Динамическое пограммирование 3: построение соединений

Для каждой пары отношений строятся все возможные комбинации методов сканирования и алгоритмов соединения, а также получаемые на выходе упорядочения.

Для каждого частичного плана вычисляется оценка стоимости.

Выбирается лучший частичный план в каждом классе эквивалентности.

Возможно, удаляется неупорядоченный частичный план.

Динамическое программирование 4: итерация

Для каждого из частичных планов, содержащих $i-1$ соединений, строятся все возможные комбинации алгоритмов соединения с отношением, еще не включенным в частичный план.

Вычисляются оценки стоимости, выбираются лучшие планы в классах эквивалентности.

Динамическое программирование 5: завершение

Среди полученных полных планов выбирается лучший.

Сокращение перебора: сначала строится полный план как продолжение лучшего частичного плана на каждой итерации.

Частичные планы, оценка которых хуже, чем оценка полученного полного плана, исключаются из рассмотрения.

Экспоненциальная сложность в худшем случае.

Оптимизация методом случайного поиска

Преобразования полного плана вместо построения частичных планов.

Пространство возможных планов превращается в граф.

Поиск: Локальные и глобальный минимумы.

Случайный поиск 2: итеративный спуск

Начиная со случайной вершины, выбирается переход, не ухудшающий стоимость, пока не будет достигнут локальный минимум.

Процесс повторяется много раз, начиная с других случайно выбранных планов.

После заданного числа итераций выбирается лучший из найденных локальных минимумов.

Случайный поиск: спуск с отступлениями

Случайное блуждание, при котором допускается не только спуск, но и подъем (с некоторой вероятностью).

Эта вероятность постепенно уменьшается с увеличением числа итераций до нуля.

Возвращается наилучший из просмотренных планов.

Случайный поиск: двухфазная оптимизация

На первой фазе выполняется итеративный спуск в течение небольшого времени.

Лучший из найденных локальных минимумов используется в качестве стартовой точки для поиска с отступлениями (на второй фазе).

Сравнение алгоритмов оптимизации

Характеристики всех методов зависят от структуры алгебраического пространства и пространства методов, а также от модели стоимости и методов оценки размеров и распределений.

Для запросов, содержащих менее 10 соединений, динамическое программирование обеспечивает лучшие результаты.

Для больших запросов обычно двухфазный случайный поиск превосходит другие методы.

Другие стратегии поиска

Полиномиальные алгоритмы используют специальный вид функции стоимости для алгоритма NL.

Комбинированные алгоритмы используют (полиномиальный) детерминированный алгоритм для поиска начальной точки случайного поиска.

Эвристические алгоритмы.

Модель стоимости: характеристики отношений

T_R количество кортежей

B_R количество блоков

Соотношение между B_R и T_R зависит от кластеризации.

$I_{R.A}$ количество различных значений атрибута

Наличие индексов

Оптимизация селекций из одного отношения

Выбирается одино из условий, по которому стоимость селекции минимальна, остальные условия проверяются просмотром отобранных кортежей.

1. Условие вида $A = c$ и отношение кластеризовано по $A \Rightarrow B_R/I_A$.

2. Условие $A\theta c, \theta \in \{<, \leq, >, \geq\}$, отношение кластеризовано по $A \Rightarrow B_R/2$.

3. Условие вида $A = c$ и имеется индекс по A (но отношение не кластеризовано) $\Rightarrow T_R/I_A$.

Оптимизация селекций 2

4. Полный просмотр отношения, хранящегося отдельно от других $\Rightarrow B_R$.

5. Отношение кластеризовано вместе с другими, но имеет отдельный индекс по атрибуту, по которому оно кластеризовано $\Rightarrow B_R$.

6. Отношение кластеризовано вместе с другими, имеется индекс по атрибуту A, условие $A|thetac \Rightarrow T_R/2$.

Оптимизация селекций 3

7. Имеется какой-нибудь индекс для отношения $\Rightarrow T_R$.

8. Полный просмотр отношения, хранящегося вместе с другими $\Rightarrow > B_R$.

Оценка стоимости прямого произведения

Число блоков для записи $R \times S$:

$$U = \frac{T_R T_S l_R}{b} + \frac{T_R T_S l_S}{b} = B_R T_S + T_R B_S.$$

Оценка снизу для стоимости: $B_R(T_S + 1) + B_S(T_R + 1)$.

Стоимость алгоритма NL:

$$B_R \left(T_S + \frac{B_S}{M - 1} \right) + B_S(T_R + 1)$$

M — количество доступных буферов в памяти.

Оценка размеров результата соединений

$R \subset D_A \times D_B, S \subset D_B \times D_C$

$p(ab \in R) = T_R/I_A I_B \; p(bc \in S) = T_S/I_B I_C$

$p(abc \in R \bowtie S) = \frac{T_R T_S}{I_A I_B^2 I_C}$

Ожидаемый размер $R \bowtie S$:

$$T_{R \bowtie S} = p(abc \in R \bowtie S) i_A I_B I_C = \frac{T_R T_S}{I_B}.$$

Стоимость операции соединения

Количество блоков, необходимых для $R \bowtie S$: $(B_R T_S + B_S T_R)/I$.

Стоимость NL: $B_R \left(\frac{T_S}{I} + \frac{B_S}{M_1}\right) + B_S \left(\frac{T_R}{I} + 1\right)$.

Merge join: $B_R \log B_R + B_S \log B_S + (B_R T_S + B_S T_R)/I$.

Hash join: $B_R + 2B_S + (B_R T_S + B_S T_R)/I$.

Неравномерность распределения значений

Неравномерность при селекции: гистограммы

```
select attribute_range, count(*) from relation
group by attribute_range;
```

Оптимизация, основанная на правилах

Для реляционной алгебры справедливы правила ассоциативности и дистрибутивности для некоторых операций.

Правила используются так, чтобы не ухудшить сложность в любом случае.

Расширяемые оптимизаторы: правила и эвристики отделены от программного кода оптимизатора.

Оптимизация в параллельных системах

Стоимость операций зависит от размещения данных

Операции могут выполняться параллельно на разном количестве процессоров

Параллельное выполнение независимых подвыражений

Распределенные системы

Узлы менее тесно связаны, чем в параллельных СУБД

Нельзя пренебрегать стоимостью сетевых взаимодействий.

Глобальный оптимизатор выбирает представление запроса на подзапросы, оптимизируемые и выполняемые локально.

Соединение результатов локальных подзапросов: операция полусоединения.

Адаптивные алгоритмы

Выполнение запроса начинается параллельно, используя различные планы (локальные минимумы).

Статистика собирается по мере выполнения запросов, планы с более лучшими результатами получают более высокий приоритеет.

Использование оперативной памяти

Стратегии освобождения буферов: LRU?

Управление буферами для иерархических структур

Оптимальное выделение буферного пула

Неэффективность LRU - пример

B-дерево с коэффициентом ветвления 100:

Уровень	количество узлов
2 (корень)	1
1 (индекс)	100
0 (данные)	10000

Выполняются запросы на выборку по точно заданному значению ключа.

Размер буферного пула = 101.

Оценки производительности: LRU

При выполнении каждого запроса выбирается по одному узлу каждого уровня (2,1,0).

Уровень	вероятность обнаружения в буфере	чтение диска
2	1	0
1	0.495	0.505
0	0.00495	0.99505
Всего		1.4951

Пример: оптимальное управление буферами

Выделяется 1 блок для уровня 2, 98 блоков для уровня 1 и 1 блок для уровня 0.

Уровень	вероятность обнаружения в буфере	чтение диска
2	1	0
1	0.98	0.02
0	0.0001	0.9999
Всего		1.0199

Оптимальное управление буферами для иерархических структур данных

Модель: Имеется s уровней, заданы вероятность доступа для каждого уровня:

$$p_1 < p_2 < \dots p_s.$$

Оптимальное выделение буферов - пропорционально вероятностям:

$$p_1 \frac{B}{p_1 + p_2 + \dots p_s}, p_2 \frac{B}{p_1 + p_2 + \dots p_s}, \dots p_s \frac{B}{p_1 + p_2 + \dots p_s}.$$

Стереотипы доступа

Последовательное сканирование: оптимально выделение 2-3 буферов.

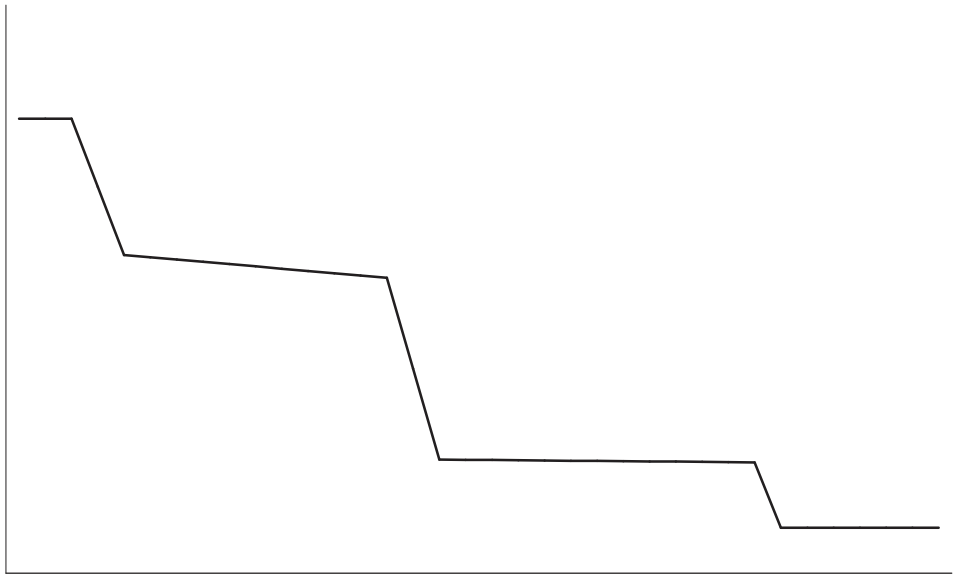
Соединение NL: 2 буфера на внутренний цикл, остальные - на внешний.

Соединение (hash join): LRU, если первое отношение не помещается в память.

Сортировка: число потоков * 2.

Эффект увеличения числа буферов

Время выполнения запроса уменьшается неравномерно. Модель: кусочно-линейная функция.



Выбор оптимального размера буферного пула

Для каждой операции - одна из “оптимальных” точек.

Количество буферов для запроса может быть вычислено на основании плана.

Проблемы:

1.Оценка оптимального плана зависит от размера буферного пула.

2. Распределение общего буферного пула между транзакциями.

Базы данных в оперативной памяти

Появляются устройства большой емкости с характеристиками, приближающимися к характеристикам оперативной памяти.

Необходим пересмотр алгоритмов индексирования и выполнения операций, а также новые модели стоимости.

Учет влияния кэширования в процессоре

Размещение данных в блоке: по строкам или по столбцам? - лучшая работа на VLDB'01.

Работа Дмитрия Шапоренкова по индексам для условий на включение множеств.

5 Параллельные базы данных

Метрики и критерии качества для параллельных СУБД

Архитектуры параллельных СУБД

Размещение данных и селекция

Параллельные алгоритмы соединения

Открытые вопросы

Основные особенности параллельных СУБД

Система использует несколько процессоров с локальной и/или общей памятью.

Данные размещаются на нескольких дисках, способных работать одновременно.

Быстрые соединения между процессорами (и дисками).

Отдельные запросы могут выполняться на нескольких процессорах параллельно.

Метрики для оценки параллельных СУБД

Конфигурация (a,b,c): a процессоров, b дисков, база данных размера c*s.

Ускорение: $\frac{\text{время ответа (1,1,1)}}{\text{время ответа (n,n,1)}}$

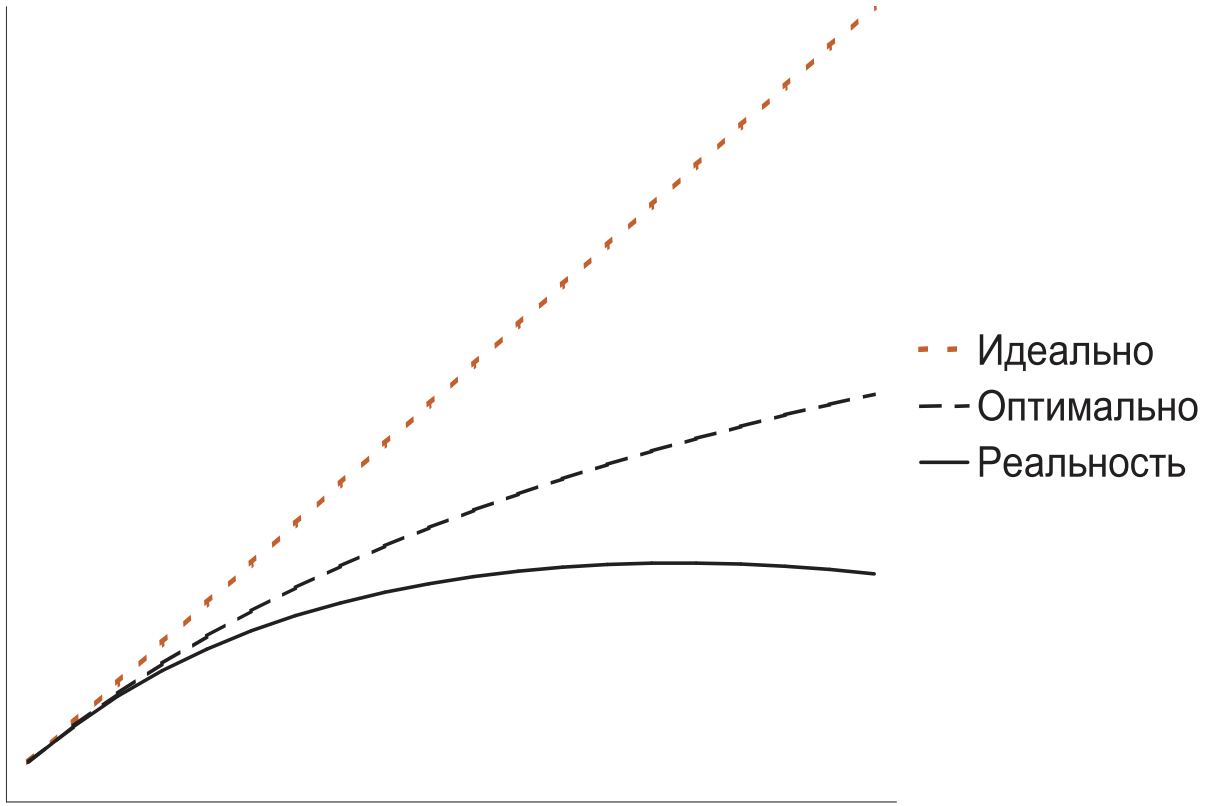
Масштабирование пропускной способности:

$$\frac{\text{пропускная способность (n,n,n)}}{\text{пропускная способность (1,1,1)}}$$

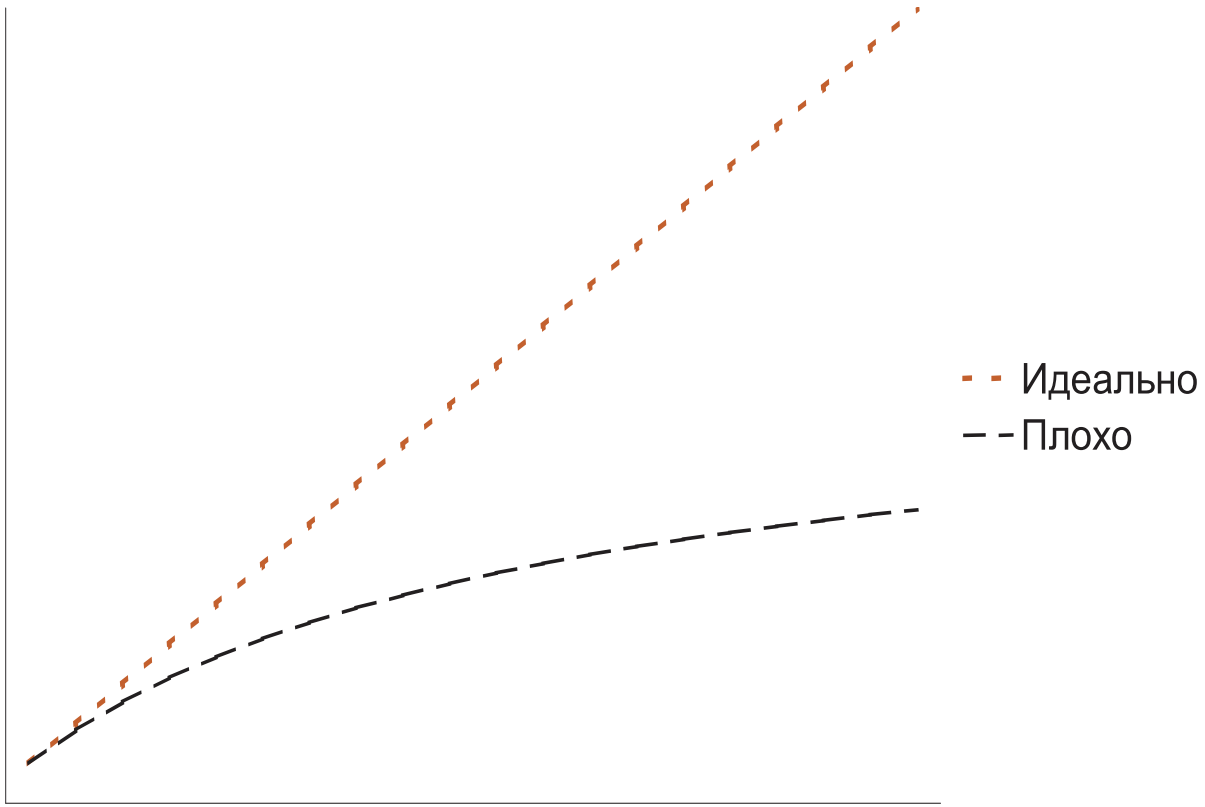
Масштабируемость времени ответа: $\frac{\text{время ответа (n,n,n)}}{\text{время ответа (1,1,1)}}$

Ускорение

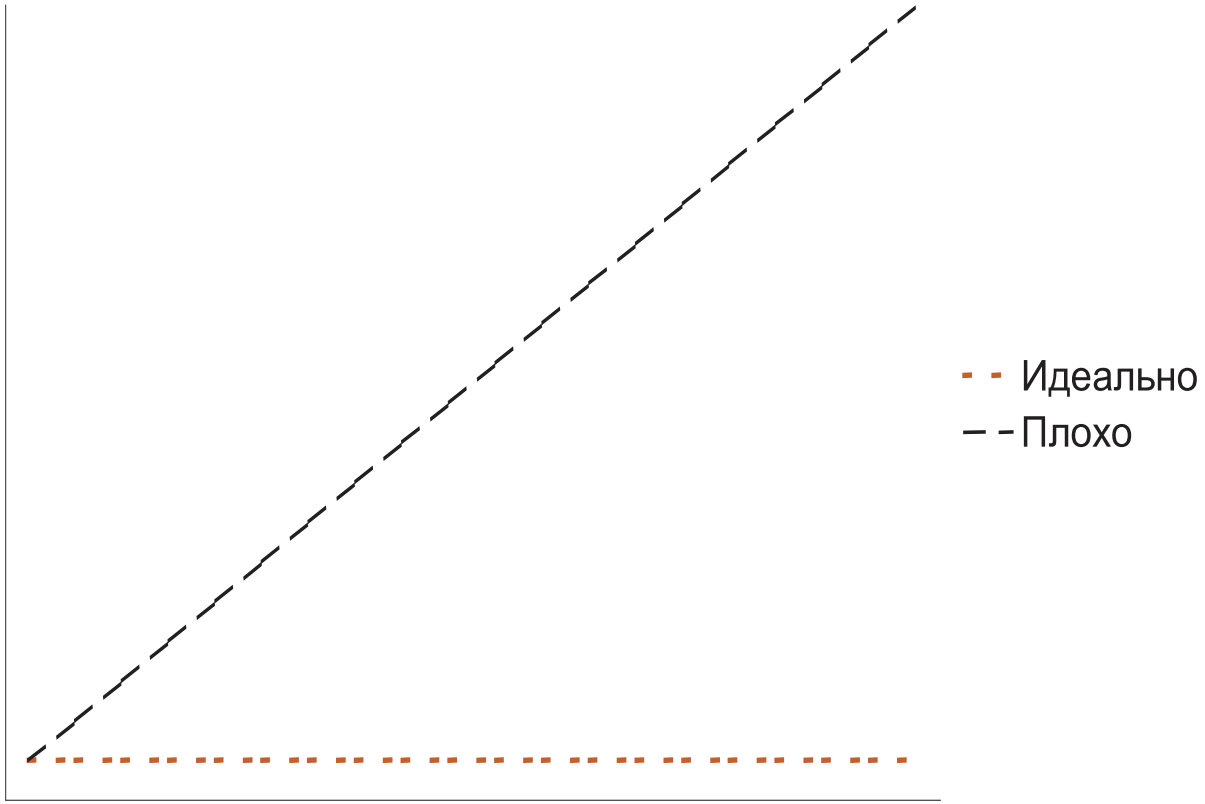
Speed-up$\leq \frac{1}{s+p/n}$



Масштабируемость пропускной способности



Масштабируемость времени ответа



Архитектуры оборудования для параллельных СУБД

SM (shared memory): Разделяемая память (это же называется SMP)

SD (shared disks): Разделяемые диски

SN (shared nothing): только сетевые взаимодействия

Системы с разделяемой памятью

Все процессоры имеют равные возможности доступа ко всем данным.

Данные направляются к операциям, их использующим.

Степень параллелизма и равномерность нагрузки на диски полностью определяется размещением данных.

Архитектура без разделения устройств

Операции пересылаются в те узлы, где хранятся данные.

Размещение данных определяет степень параллелизма и баланс нагрузки как по вводу/выводу, так и по процессору.

Архитектура с разделяемыми дисками

Используется как пересылка данных, так и пересылка операций.

Размещение данных определяет степень параллелизма и баланс ввода/вывода.

Пересылка операций определяет баланс процессорной загрузки.

Виды параллелизма

Intertransaction

Intratransaction/interquery

Intraquery/interoperator

Intraoperator

Размещение данных: декластеризация таблиц хешированием

Узел для размещения строки таблицы определяется функцией хеширования.

Inter-query: Параллельный доступ при поиске по значению атрибута декластеризации.

Intra-query: параллельный полный просмотр таблицы

Размещение данных: декластеризация таблиц по диапазонам

Область значений атрибута представляется суммой диапазонов значений, для каждого выделяется отдельный раздел таблицы (на отдельном диске).

Inter-query: точные значения и малые диапазоны атрибута

Intra-query: полный просмотр и большие диапазоны.

Декластеризация таблиц: чередующиеся диапазоны (interleaving)

Соседние диапазоны малого размера размещаются в разных разделах, разделы назначаются циклически.

Необходимо оглавление для поиска нужного раздела.

Допускает настройку степени параллелизма для выборки по диапазону среднего размера.

Декластеризация индексов

Локальные индексы (для каждого раздела). Используются для вторичных неуникальных индексов.

Уникальные индексы должны быть реализованы как отдельные декластеризованные таблицы.

Глобальные индексы сами должны быть декластеризованы.

Распределение на уровне блоков

Распределяются физические блоки, а не логические записи.

Небольшие группы соседних блоков распределяются между дисками, которые используются циклически.

Степень параллелизма существенно зависит от количества соседних блоков, размещаемых вместе: слишком большие фрагменты уменьшают степень параллелизма.

Параллельные алгоритмы соединения

Для вычисления $R \bowtie S$ таблицы R и S разбиваются на фрагменты и затем параллельно выполняются локальные операции соединения всех фрагментов.

Необходимо использовать существующую декластеризацию исходных таблиц.

Фазы распределения и соединения могут выполняться параллельно (pipelining).

Параллельное соединение: вложенные циклы

Фаза распределения: все кортежи R рассылаются во все узлы, в которых хранится S.

Фаза соединения: Вычисляются локальные соединения в узлах S.

Если S декластеризовано по атрибуту соединения, на фазе распределения можно использовать эту декластеризацию.

Параллельный hash-join

На фазе распределения R перераспределяется с использованием функции хеширования.

При размещении кортежей R в каждом узле строится локальная таблица хеширования.

Затем S распределяется с помощью той же функции хеширования,

параллельно выполняется соединение в каждом узле.

Если локальные таблицы хеширования для R не помещаются в память, необходимо использовать локальный hash-join.

Выравнивание нагрузки

Неравномерная нагрузка возникает вследствие неравномерного распределения значений атрибута, неравномерного распределения при хешировании, неравномерности распределения в генерируемом результате операции.

Меры по выравниванию загрузки:

Перед фазой декластеризации вычислить распределения и использовать эту информацию для назначения узлов при хешировании.

Перераспределять группы, когда перекос данных обнаружен.

Параллельная сортировка

Выполняются локальные сортировки в каждом узле, в котором находится исходная таблица, затем полученные упорядоченные фрагменты декластерзуются по узлам, в которых будет размещен результат, с одновременным слиянием в этих узлах.

Трубопроводная сортировка: параллельно выполняются несколько проходов сортировки, при достаточном количестве процессоров сортировка выполняется за линейное время.

Паралеллизм между операциями в дереве запроса

Основан на использовании pipeline.

Для операций соединения существенна последовательность выполнения операций.

$$(R_1 \bowtie (R_2 \bowtie (R_3 \ldots \bowtie (R_{n-1} \bowtie R_n) \ldots)$$

$$(R_1 \bowtie R_2) \bowtie R_3) \ldots \bowtie R_{n-1}) \bowtie R_n) \ldots)$$

В первом случае фаза распределения для всех операций соединения может выполняться параллельно, и затем параллельно все фазы соединения.

Темы, не затронутые в этом разделе

Параллельные объектно-ориентированные СУБД

Выполнение рекурсивных запросов в параллельных системах

Отказоустойчивость

Открытые проблемы

Проектирование схемы хранения для параллельных систем: выбор стратегий размещения данных, динамические методы размещения, распределенные индексные структуры.

Оптимизация запросов: учет интегральных критериев производительности в функциях стоимости, учет доступности ресурсов.

Гибридные архитектуры систем.

6 Специальные типы данных

Индексирование в объектных системах

Индексы очень большой размерности

Индексирование текстов и информационный поиск

Объектные системы баз данных: мотивации

Различие в уровне языков запросов и языков программирования: Impedance mismatch.

Отсутствие пользовательских типов данных в ранних реляционных СУБД.

Потребности новых приложений: обработка сложных объектов.

Необходимость навигационных операций.

Классы объектных систем баз данных

Языки программирования баз данных и объектные системы с постоянно хранимыми объектами: долговременная устойчивая (persistent) виртуальная память большого размера

Объектные базы данных: объектные языки запросов

Объектно-реляционные базы данных: классы объектов в качестве доменов коллекций, расширенные реляционная алгебра и исчисление.

Проблемы организации хранения в постоянной виртуальной памяти

Эффективность систем критически зависит от кластеризации взаимосвязанных объектов на одной странице

Невозможность сборки мусора влечет необходимость учета количества ссылок

Грубые решения: реализуется функциональность, не поддержанная специальными структурами хранения (Microsoft).

Проблемы организации хранения в постоянной виртуальной памяти 2

Генерация указателей: абстрактные или адресные.

Перемещаемость объектов: необходимость косвенной адресации или хеширования.

Проблема преобразования указателей (swizzling): указатели на объекты, находящиеся в памяти, могут быть преобразованы во внутренние указатели.

Стандарт объектных баз данных ODMG

Cattell, R.G.G., et al. The Object Database Standard: ODMG 3.0. Morgan-Kaufmann publishers. 2000.

Предыдущие версии: ODMG-93, ODMG 2.0 (1997).

ODMG = Object Data management Group

Описывает языки ODL, OQL, но модель данных описывается в ODL

Альтернатива SQL:1999.

Расширения, необходимые для реализации объектно-реляционных систем

Новые типы коллекций: новые алгебраические операции

Сложные структуры: вложенные коллекции

Сочетание навигационных и реляционных операций: пути.

Наследование: хранение и индексирование

Индексирование сложных объектов

Операции над новыми типами коллекций

Необходимы варианты операций для всех сочетаний типов аргументов.

Усложняется работа оптимизатора вследствие расширения пространства возможных планов.

Реализация алгоритмов не составляет проблемы, поскольку удаление дубликаторов и сортировка выполняются как отдельные операции.

Сложные структуры данных – вложенные отношения

Non-First Normal Form - NFNF NF^2

В модели вложенных отношений допускаются отношения в качестве значений атрибутов

Обеспечивается логическая группировка взаимосвязанных данных как в иерархической модели данных

Новые операции: Nest, Unnest

Расширенная реляционная алгебра

Хранение вложенных коллекций

Реляционное представление: вложенные коллекции хранятся в отдельных областях памяти (с дополнительным ключом кортежа содержащего вложенную коллекцию).

Преимущество: объекты верхних уровней кластеризованы.

Иерархическое представление: вложенные коллекции внутри кортежей родительской коллекции.

Преимущество: дерево вложенных объектов кластеризовано.

Поиск по путям: nested indexes

Классы $obj_1, \ldots obj_s$ с атрибутами $A_1, A_2, \ldots A_s$ соответственно, тип A_i - указатель на $obj_{i+1} 1 \leq i \leq s-1$.

Критерии в реляционных выражениях (селекции и соединений) могут содержать условия вида:

$$Obj_1.A_1.A_2.\ldots_p \theta \text{const or expr}$$

Вложенные индексы: обычный индекс (например, B-дерево) по значениям A_p, включающий ссылки на объекты Obj.

Поиск по путям: отношения путей доступа

Для часто используемых путей строятся отношения со схемой:

$$A_1, A_2, \ldots A_p$$

При наличии общих участков в разных путях возможна нормализация.

Для каждого из отношений строятся индексы по первому и по последнему атрибутам для ускорения операций соединения.

Недостаток: очень сложные обновления.

Навигационный поиск: скелеты БД

Скелет: для каждого объекта база данных содержит его идентификатор и значения атрибутов, являющиеся ссылками на другие объекты.

Для баз данных, содержащих объекты большого размера, скелет занимает значительно меньше места и может быть полностью загружен в память.

При выполнении запросов все навигационные операции могут использовать скелет, обращения к базе данных нужны только для доступа к другим атрибутам.

Индексы для сложных объектов: сравнение

При малом количестве путей вложенные индексы являются наиболее быстрыми.

Отношения путей доступа обеспечивают дополнительную функциональность (поиск по частичному пути).

Скелеты эффективны при наличии атрибутов большого размера (образы, звук, видео), но не масштабируемы.

Наследование - организация хранения

Выделение памяти для объектов: общее пространство или отдельно для каждого типа.

При выделении для каждого типа подтипы могут храниться вместе с типом или как отдельная коллекция.

Для подтипа могут храниться только дополнительные атрибуты.

Кластеризация типов и подтипов в одном кластере.

Наследование: индексы

Проблема: поиск по значению атрибута в подтипе при наличии индекса по тому же атрибуту для супертипа.

HB$^+$-деревья: B$^+$-дерево, вместе со ссылками на объекты в листьях указываются их типы.

Для каждого подтипа строится отдельный индекс B$^+$-дерева, содержащий ссылки только на те листья, в которых есть объекты этого подтипа.

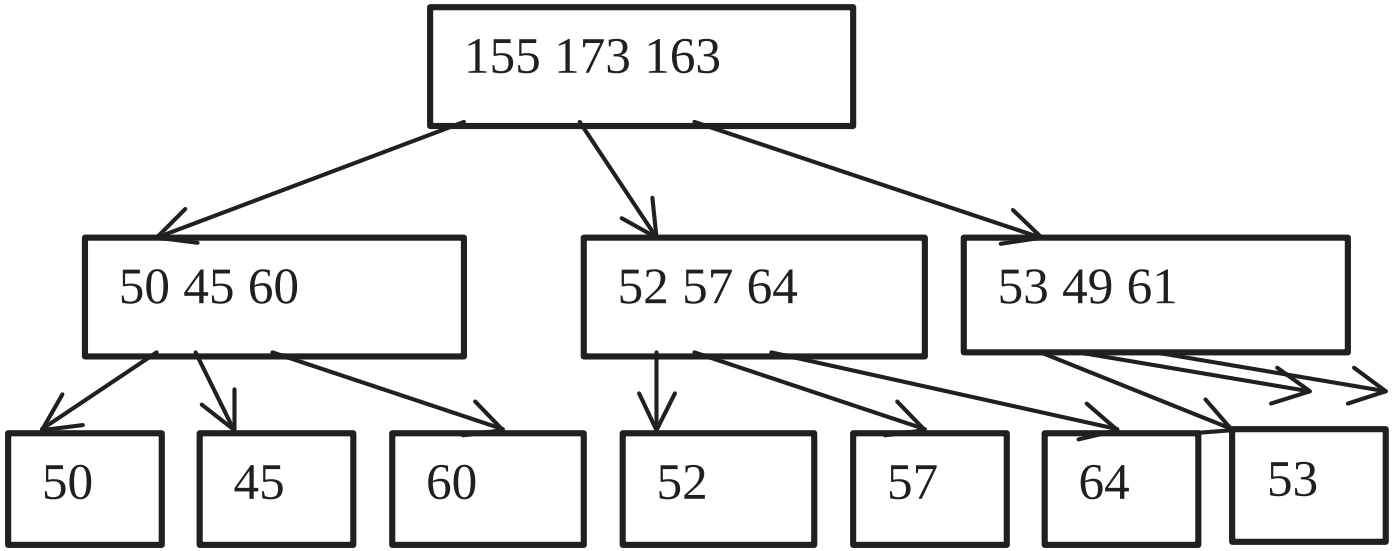
Хранение больших объектов

Объекты большого размера, структура которых не определена для СУБД: тексты, образы, звук,...

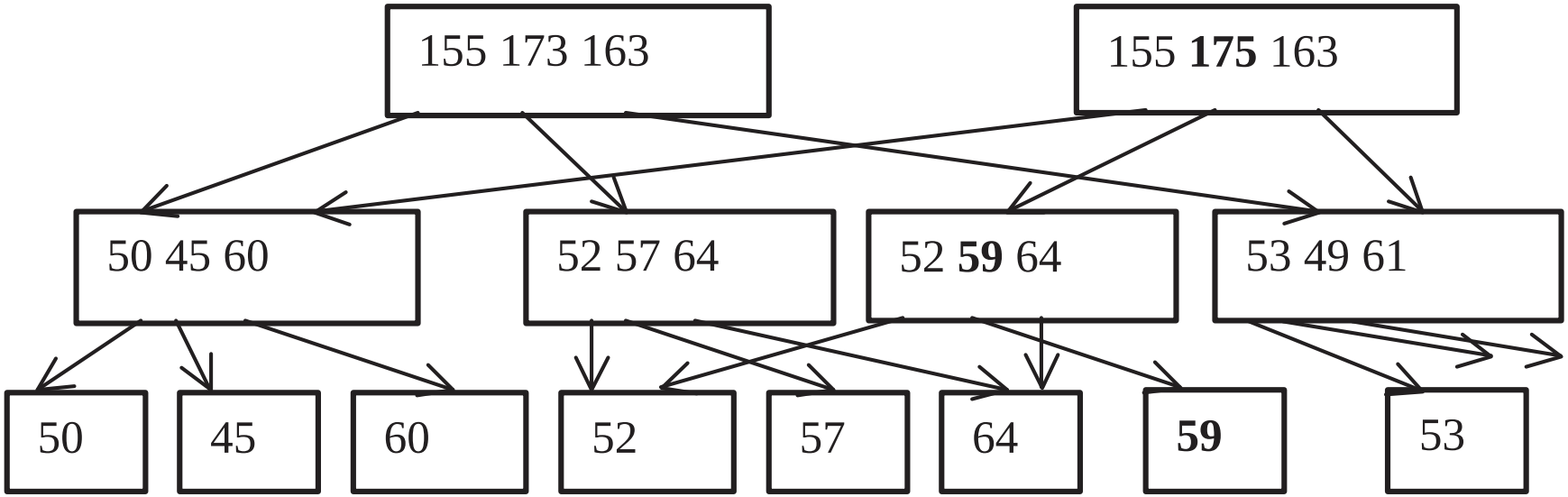
Требуется:

- Быстрый доступ к произвольным частям объекта по смещению от начала.
- Возможность частичной модификации объекта.
- Поддержка нескольких версий объекта.

Большие объекты: структура данных



Большие объекты: версии



Рекурсивные запросы

Дедуктивные базы данных состоят из:

- экстенсионала (фактов, представленных отношениями)
- интенсионала (правил вывода, определяющих неявные отношения).

Пример рекурсивных правил:

```
a(X,Y) :- p(X,Y).
a(X,Y) :- a(X,Z),p(Z,Y).
```

Рекурсивные запросы

Рекурсивные запросы не могут быть выражены в терминах операций реляционной алгебры.

Рекурсивные запросы необходимы для обработки деревьев неопределенной глубины.

Возможна запись рекурсивных запросов в SQL:1999.

Для вычисления рекурсивных запросов необходимо ввести операцию транзитивного замыкания для бинарных отношений.

Вычисление транзитивного замыкания: вложенные циклы

```
A := P;
for z in D
   for x in D
      for y in D
         if P(x,z) and p(z,y) then insert into A values(x,y)
         end
      end
   end
```

Неэффективно, так как не используется структура блоков базы данных.

Вычисление сверху вниз: обход вглубь

Это метод, используемый в языке Prolog.

Может зацикливаться на некоторых наборах данных.

Пример:

```
p(m,n)
p(n,m)
```

Вычисление снизу вверх

$$A_0 = \emptyset$$

$$A_i = A_{i-1} \bowtie P \cup P$$

Процесс останавливается, когда $A_i - A_{i-1} = \emptyset$.

Невозможно использовать дополнительные условия селекции на ранних фазах вычисления.

Метод магических множеств

Первый этап: псевдовычислением сверху вниз строятся дополнительные ограничивающие предикаты.

Второй этап: фактическое вычисление снизу вверх использует построенные предикаты для сокращения размера промежуточного результата.

Данные очень большой размерности

Возникают в многосредовых приложениях.

Структура файла (например, кадры и фреймы видеофильма) не отражает содержание.

Ручное аннотирование или автоматическое выделение характеристик (features).

Набор характеристик представляет собой вектор большой размерности.

Типичные запросы: близость (подобие) объектов.

Индексы очень большой размерности

k-d- и k-d-B-деревья.

Поведение R-деревьев при увеличении размерности.

Любой метод, основанный на деревьях, с ростом размерности становится хуже линейного просмотра.

Улучшение линейного поиска: сигнатуры.

Индексирование текстов и информационный поиск

Задача информационного поиска: обнаружение пертинентных документов.

Задача, решаемая поисковыми системами: нахождение релевантных документов.

Модели поисковых систем: булевская, векторная, вероятностная.

Лингвистическая обработка документов и запросов.

Критерии качества: полнота и точность.

Инвертированные файлы

Поддерживают булевскую модель поиска.

Для каждого (лингвистически обработанного) слова строится список (номеров) документов, содержащих это слово.

Полный индекс занимает больше места, чем индексируемые документы.

Индексирование с точностью до документа, абзаца, предложения.

Часто встречающиеся и редко встречающиеся слова.

Сжатие инвертированных файлов

Словари невелики по объему и могут размещаться в оперативной памяти.

Необходимо сжатие инвертированных списков, удовлетворяющее условиям:

- невысокая вычислительная сложность,
- возможность выполнения логических операций над списками “на лету”.

Сжатие инвертированных списков 2

Документы представляются порядковыми номерами.

Инвертированный список для каждого слова представляет собой упорядоченную последовательность номеров документов (или разности соседних номеров).

Кодирование чисел в инвертированных списках

0	01	0
1	01	1
2	001	10
3	001	11
4	0001	100
5	0001	101
6	0001	110
7	0001	111
8	00001	1000

0	01	0	0
1	01	0	1
2	01	1	10
3	01	1	11
4	001	10	100
5	001	10	101
6	001	10	110
7	001	10	111
8	001	11	1000

Варианты сжатых инвертированных списков

Каждое вхождение в инвертированный список может состоять из:

- только номера документа,
- номера документа и количества вхождений в документ,
- номера документа, количества вхождений и номеров позиций слова в документе.

Размер индекса: 5–25% от объема документов.

Кодирование наложением (сигнатурные файлы)

Каждый документ представляется битовой шкалой длины s.

Шкала для слова вычисляется с помощью функции хеширования, вырабатывающей шкалу длины s и содержащей ровно m единичных битов.

Сигнатура документа (или запроса) - дизъюнкция входящих в него слов.

Основной факт: маска документа, релевантного запросу, мажорирует маску запроса.

Недостатки метода кодирования наложением

Неизбежен шум (выборка нерелевантных документов).

Время поиска линейно зависит от объема данных.

В целом эти индексы уступают инвертированным спискам.

Векторные модели информационного поиска

Документ (или запрос) представляется вектором, каждая компонента которого является весом слова, встречающегося в базе данных.

Эти векторы необходимо нормализовать.

Основная гипотеза: Близкие по содержанию документы описываются векторами, близкими в некоторой метрике многомерного пространства.

Многочисленные и многолетние эксперименты показывают, что эта гипотеза правдоподобна.

Информационный поиск — тема отдельного курса.

Другие темы

В этом курсе (в этом году) не рассматривались многие важные темы, в том числе:

- Хранение и индексирование слабоструктурированных данных и XML.
- Оптимизация запросов в неоднородных автономных системах и запросов над WWW.
- OLAP.
- Многое другое.

Имеется большой простор для исследовательской работы.