Обработка и исполнение запросов в СУБД (Лекция 5)

Классические системы: принципы построения распределенных СУБД

v2

Георгий Чернышев

Академический Университет

chernishev@gmail.com

19 октября 2016 г.

РСУБД: что нового по сравнению с централизованной?

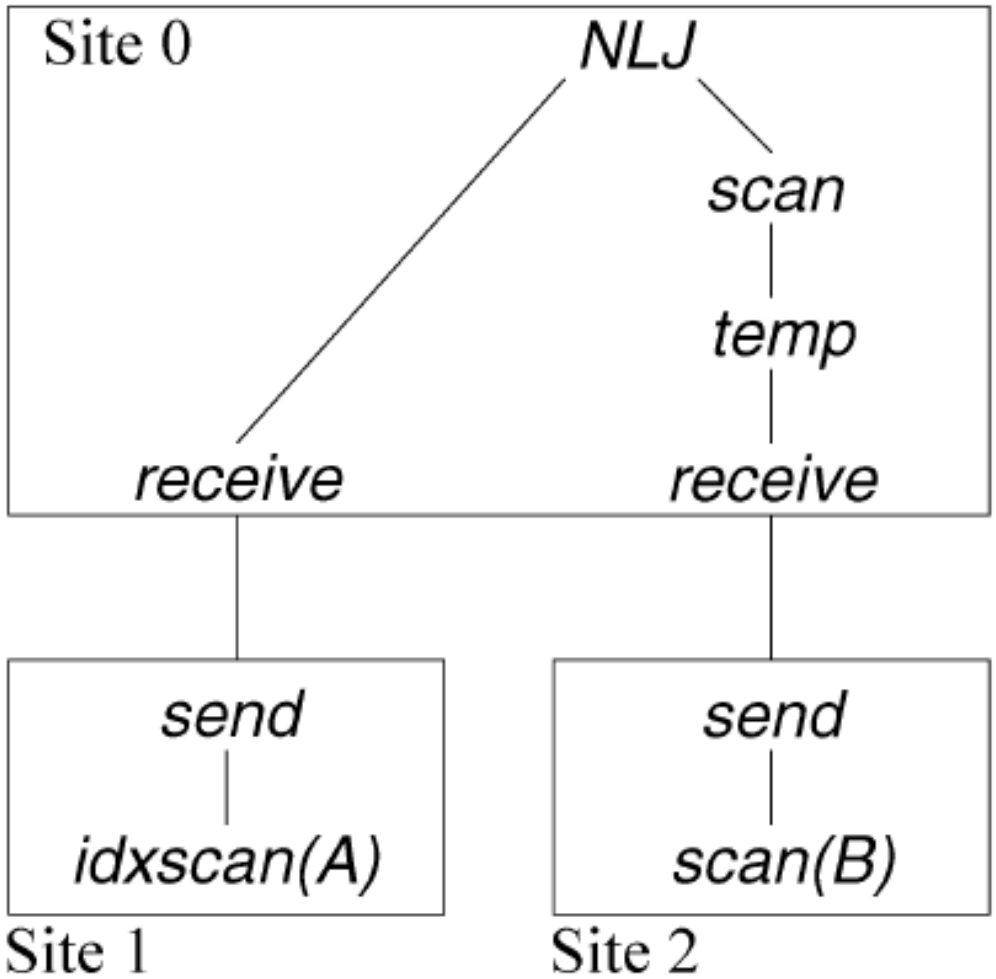


Fig. 2. Example query evaluation plan.

[1]

[1] Изображение взято из [Kossmann, 2000]

Распределенная оптимизация

- Отношения могут быть реплицированы;
- Стандартный алгоритм динамического программирования (лекция 2), но теперь не просто access methods, а $scan(A, S_1)$ и $scan(A, S_2)$;
- Концепция interesting sites, аналог interesting orders;
- Выигрыш от bushy plans выше в распределенных системах.

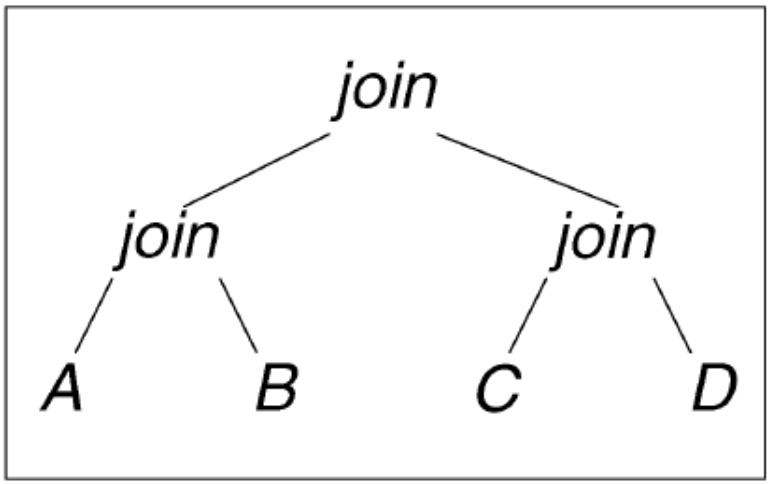
$\longrightarrow$ оптимизация сложнее нежели в централизованных системах

Стоимостные модели для распределенной оптимизации

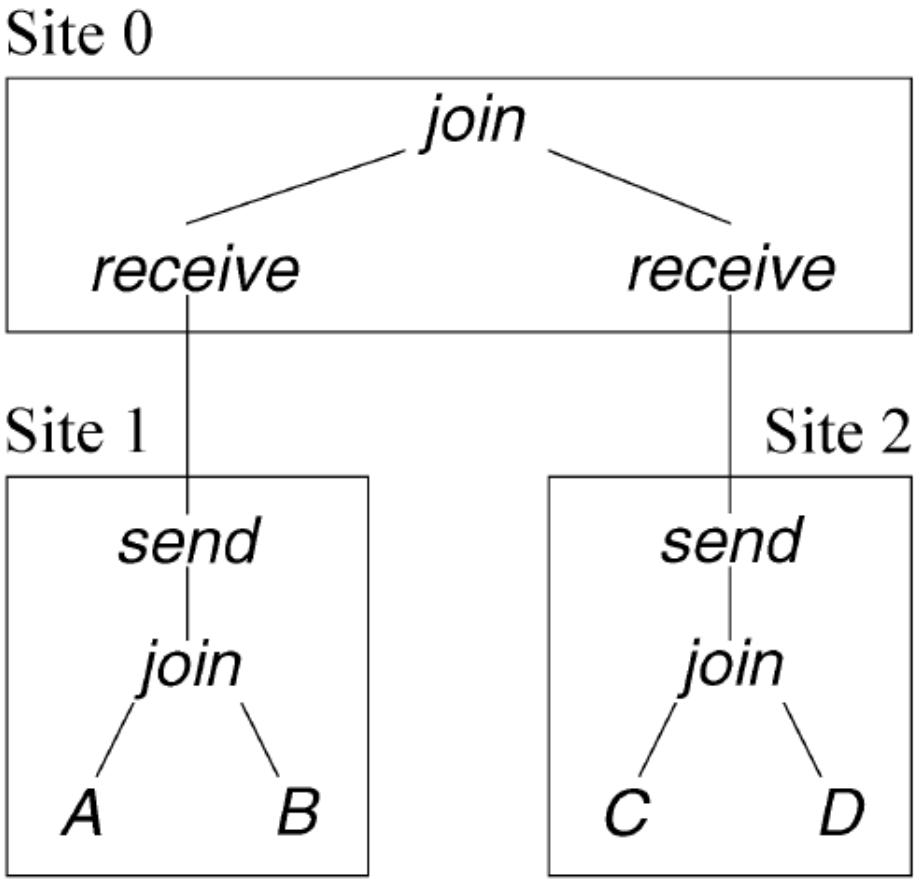
- Модель потребления ресурсов:
  - Централизованная система: CPU + disk I/O;
  - Распределенная: ... + network I/O + packing/unpacking;
  - Линейная комбинация с весами;
- Модель времени ответа:
  - Иногда нужен быстрый ответ $\longrightarrow$ нужен внутризапросный параллелизм;
  - Модель учитывает внутризапросный параллелизм и pipelining (aka неблокирующие операторы, лекция 1).

Конкретные примеры есть в 8 главе [Özsu and Valduriez, 2009].

Две модели: пример



Minimum Resource Consumption



Minimum Response Time

Fig. 4. Example plans: total resource consumption vs. response time. [2]

[2] Изображение взято из [Kossmann, 2000]

Модель времени ответа “на пальцах”

1. Вычисляем сколько потребит каждый оператор в отдельности;
2. Вычисляется использование разделяемого ресурса всеми операторами работающими параллельно. Это делается для каждого ресурса.
   - Пример: сеть. Пропускная способность сети делится на объем переданных данных всеми операторами.
3. Время ответа группы операторов вычисляется как максимум по потреблению ресурсов отдельных операторов.

Пример

Предположим:

1. Все соединения во всех планах работают параллельно (поточно или нет);
2. Каждое соединение стоит 200 секунд работы CPU и нет disk I/O;
3. У сети нет задержек, доставка $A \bowtie B$ и $C \bowtie D$ стоят 130 секунд работы сети;
4. Посылка и получение данных в/из сети ничего не стоит;
5. Чтение всех таблиц ничего не стоит;

Пример, продолжение

Тогда:

1. Время работы левого плана будет 600 секунд (CPU только);
2. Время работы правого плана будет $max(130 + 130, 200) = 260$;

Всё хорошо, однако:

- не учитывается scheduling и конкуренция за ресурс: что будет если на узел оптимизатор бросит сразу много операторов из разных запросов?

Преимущество: модель дешева в использовании.

Как исполнять запросы? I

Набор приемов для построения (минимально) эффективной РСУБД:

- Поблочная передача данных (нужно настраивать под размер сообщения, учет алгоритма Нагеля и прочее);
- Оптимизация мультикастов: выбор оптимального маршрута передачи;
- Уместное использование многопоточности:

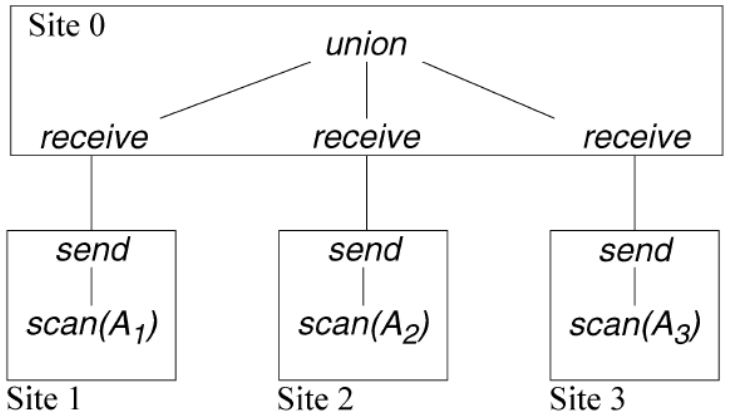


Fig. 5. Example union plan.

3

Как исполнять запросы? II

- Неуместное использование многопоточности:

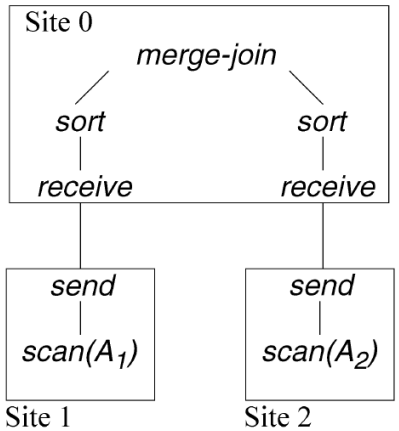


Fig. 6. Example join plan.

[4]

- Горизонтальное фрагментирование + репликация:
  Пусть $A = A_1 \cup A_2$, надо $A \bowtie B$
  - Тогда: $(A_1 \cup A_2) \bowtie B$ или $(A_1 \bowtie B) \cup (A_2 \bowtie B)$
  - Иногда можно и так: $((A_1 \cup A_2) \bowtie B) \cup (A_3 \bowtie B)$

[3] Изображение взято из [Kossmann, 2000]

[4] Изображение взято из [Kossmann, 2000]

Как исполнять запросы? III

- Попарно не пересекающиеся фрагменты отношений:

$$(Emp_1 \cup Emp_2 ... \cup EMP_n) \bowtie (Dept_1 \cup Dept_2 ... \cup Dept_n) =$$

$$(Emp_1 \bowtie Dept_1) \cup (Emp_2 \bowtie Dept_2) ... \cup (Emp_n \bowtie Dept_n)$$

- Другие реализации параллельного соединения [Taniar et al., 2008].
- Трюк с полусоединением: таблицы A и B на разных узлах

$$A \bowtie B = A \bowtie (B \ltimes \pi(A))$$

  На самом деле всё не так просто, возможных планов с полусоединением — много, см. 8 главу [Özsu and Valduriez, 2009].

- Double-pipelined hash join: борьба с неравномерностью нагрузки.

Как исполнять запросы? IV

- Оператор stop для SELECT TOP

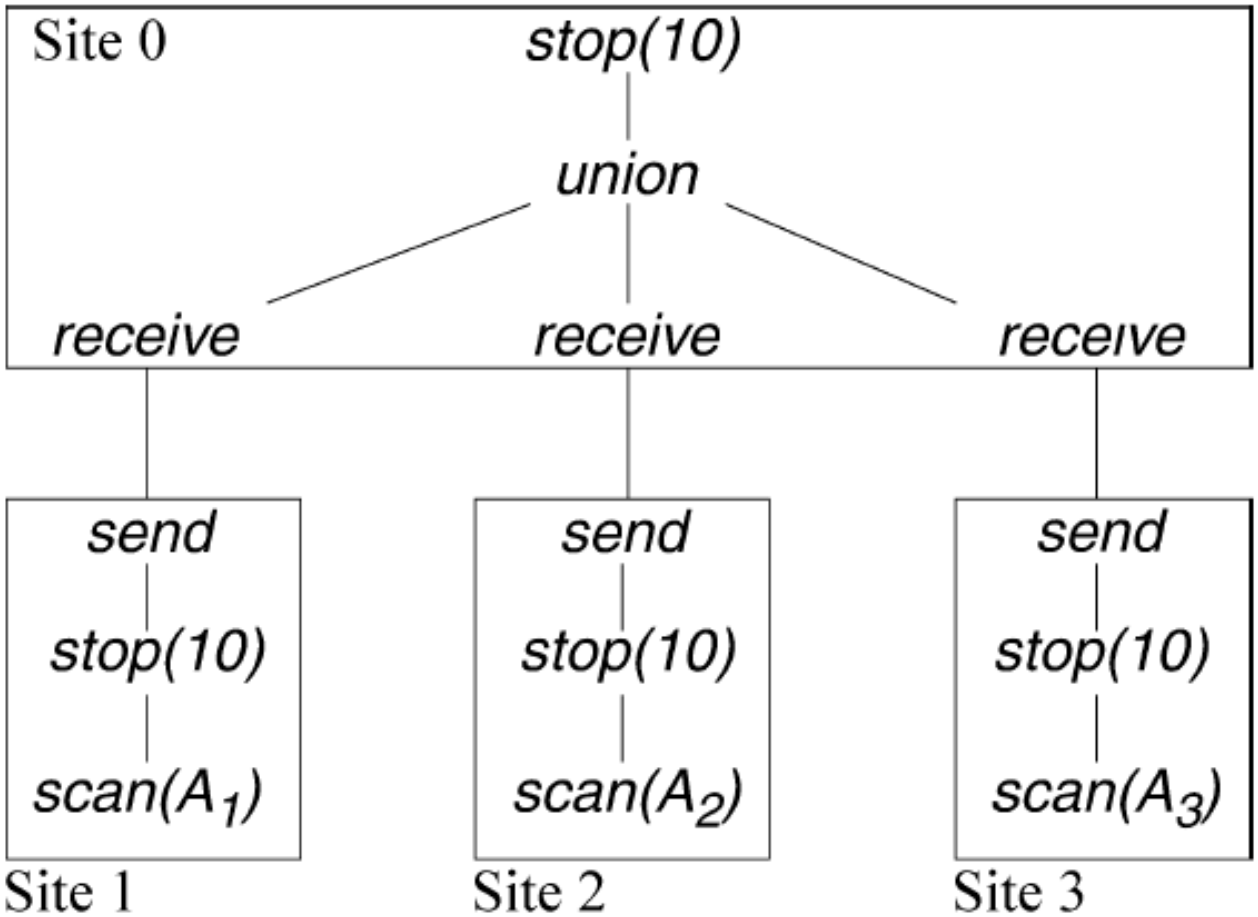


Fig. 7. Example plan for a top N query.

5

[5] Изображение взято из [Kossmann, 2000]

Как исполнять запросы? V

- Мультимедиа РСУБД, пусть надо выполнить TOP(N) и известно что функция оценки — монотонная функция

  например, найти TOP 10 от f = min{score(voice), score(looks)} тогда работает такой алгоритм:
  - Последовательно просим top от voice и looks, пока пересечение не будет иметь 10 объектов;
  - Вычисляем ранжирующую функцию f для тех птиц, что не входят в пересечение и выдаем результат.

Монотонность: если $s_1(a) < s_1(b)$ и $s_2(a) < s_2(b)$ означает $f(s_1(a), s_2(a)) < f(s_1(b), s_2(b))$.

Клиент-серверные системы

Система где разделяются типы машин (см. предыдущую лекцию): обслуживающие запросы и хранящие данные.

Основные вопросы:

- Выполнять запрос на машине где находятся данные или же на клиенте?
- Как использовать кеширование на клиентах?

Выполнение запросов в клиент-серверных системах

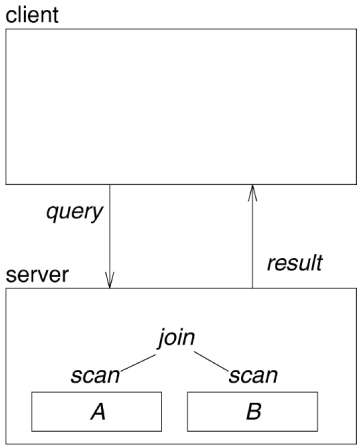


Fig. 8. Query shipping.

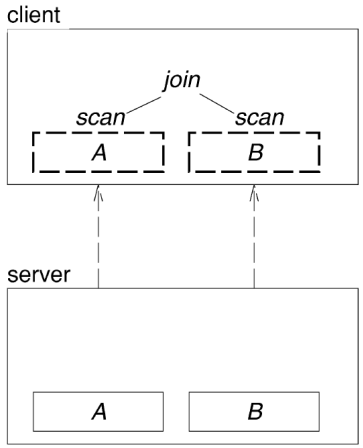


Fig. 9. Data shipping.

[6]

Стратегии:

- Миграция данных;
- Миграция запроса;
- Гибридная.

[6] Изображение взято из [Kossmann, 2000]

Что лучше?

- Зависит от машин: каждая из схем имеет свою нишу;
- В гибридных оптимизация сложнее;
- Иногда выгодно игнорировать кеши на клиентах: две выборки, соединение и быстрая сеть;
- Иногда выгодно “вернуть” (промежуточные) данные на сервер: быстрая сеть, сервер эффективно обрабатывает соединения;
- Маленькие апдейты хранить на клиентах, а большие на серверах.

Оптимизация в РСУБД I

Выбор места выполнения оператора:

Table I. Site Selection Options for Data, Query, and Hybrid Shipping

	Data Shipping	Query Shipping	Hybrid Shipping
display	client	client	client
update	client	server	client or server
binary operators (e.g., join)	consumer (i.e., client)	producer of left or right input	consumer or producer of left or right input
unary operators (e.g., sort, group-by)	consumer (i.e., client)	producer	consumer or producer
scan	client	server	client or server

[7]

Аннотации транслируются в адреса машин, затем им “раздается” код.

Если есть репликация то можно повыбирать.

[7] Изображение взято из [Kossmann, 2000]

Оптимизация в РСУБД II

Когда и где оптимизировать? Мнений много, некоторые мысли:

- Разбор и перезапись запроса лучше делать на клиенте: разгружаем сервер;
- Оптимизация на сервере: спрашивать состояние vs угадывать состояние;
- Подходы к оптимизации запросов: сохраненные планы запросов, набор планов, оптимизация на лету, реоптимизация, декомпозиция запроса, ...
- Двухшаговая оптимизация — самый популярный подход:
  1. абстрактный план: access methods, порядок соединений;
  2. прямо перед выполнением трансформировать план и выбирать узлы.

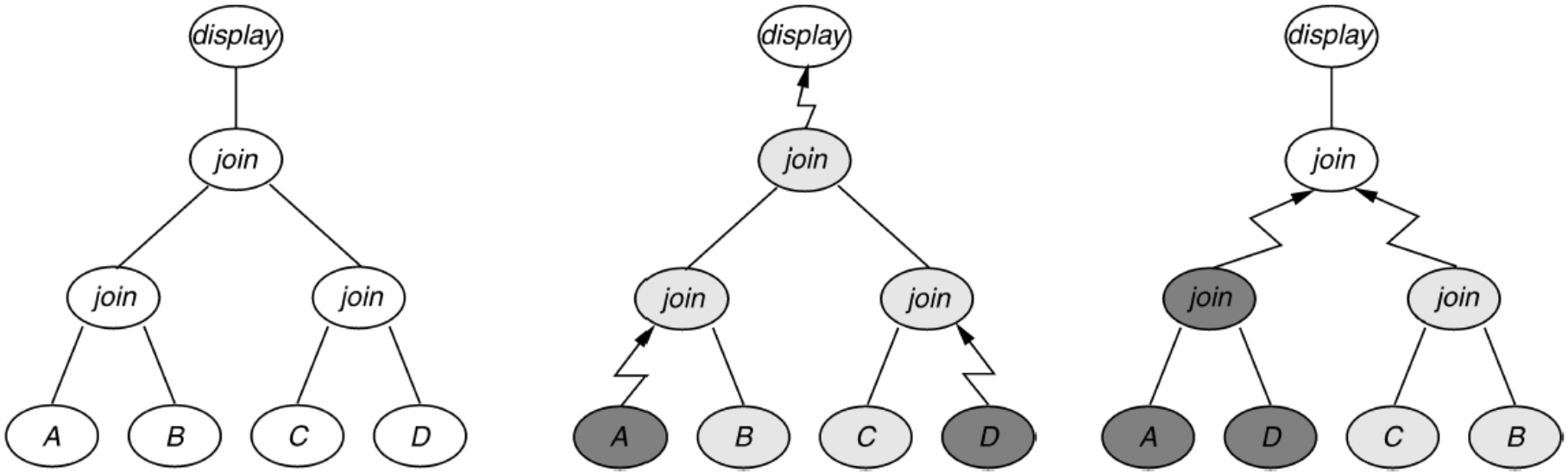
Свойства двухшагового подхода

Плюсы:

- Каждый шаг — дешев;
- Второй шаг позволяет делать load-balancing;
- Второй шаг позволяет использовать кеширование.

Минусы:

- Неоптимальные планы с точки зрения сетевой стоимости.



(a) 2-Step Plan at Compile Time (b) 2-Step Plan at Run Time (c) Optimal Plan

Fig. 11. Increased communication cost due to two-step optimization.

Исполнение запросов в гетерогенных СУБД

Гетерогенные РСУБД — РСУБД в которых каждый узел независимая СУБД, возможно различная, работающая на различном железе.

Самые острые вопросы:

- Как эффективно использовать возможности участников?
- Как решать вопрос семантической гетерогенности?

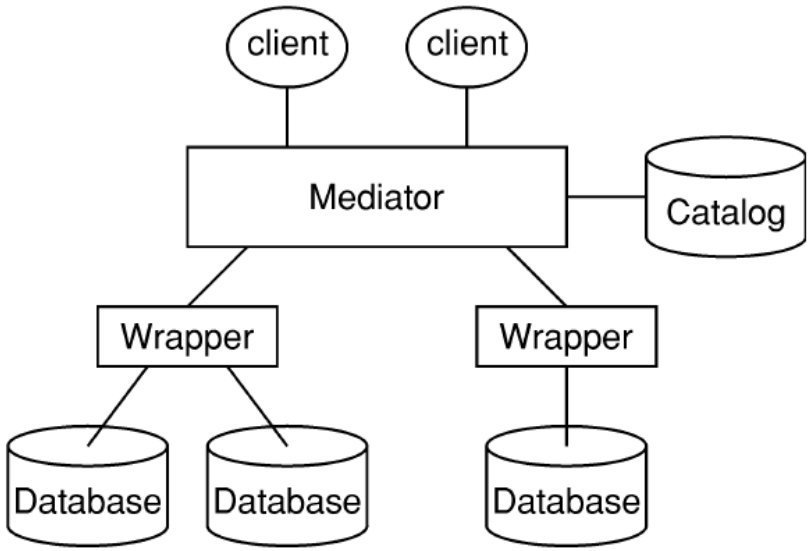


Fig. 12. Wrapper architecture of heterogeneous databases.

Описание возможностей участников

- Описать возможности участников как views, хранить в каталоге. Гибок, но сложно реализовать;
- Capability records — КСГ-грамматика, описываем возможнсти запросов. Используем их в генерации планов, нужны новые алгоритмы стоимостной и эвристической оптимизации;
- Правила обхода (enumeration rules) — подаются в оптимизатор.

Правила обхода

- Идея: каждая обертка дает наружу набор правил (в виде функций) как с ней обращаться, что она умеет делать;
- Применяем стандартный алгоритм динамического программирования;
- Правила обхода (enumeration rules) — подаются в оптимизатор, получаем промежуточное представление, подаем его участникам, те транслируют в SQL и исполняют.

Пример

$\texttt{plan\_access}(T, C, P) = \textbf{R\_Scan}(T, C, P, ds(T))$

$ds(T)$ returns the ID of the relational component database that stores T.

Fig. 13. Access plan enumeration rule for relational component databases.

$\texttt{plan\_join}(S_1, S_2, P) = \textbf{R\_Join}(S_1, S_2, P)$

Condition: $S_1.Site = S_2.Site$

Fig. 14. Join plan enumeration rule for relational component databases.

```
SELECT e.name, e.salary, d.budget
FROM Emp e, Dept d
WHERE e.salary > 100,000 AND
  e.works_in = d.dno;
```

```
plan_access(Emp, {salary,works_in,name},
      {salary > 100,000}) =
      R_Scan(Emp, {salary,works_in,name},
      {salary > 100,000}, D1)
plan_access(Dept, {dno,budget}, {}) =
      R_Scan(Dept, {dno,budget}, {}, D1)
```

```
R_Join(R_Scan(Emp,{salary,works_in,name},
     {salary > 100,000}, D1)
        R_Scan(Dept, {dno,budget}, {}, D1)
     {Emp.works_in = Dept.dno})
```

10

[10] Изображение взято из [Kossmann, 2000]

Плюсы подхода

- Работает со стандартным стеком БД-технологий;
- Гибкость;
- Легко реализовывать;
- Итеративность разработки;
- Легкость добавления правил.

Как получать статистику?

- Калибровка: generic стоимостная модель

$$n * c$$

ищем c с помощью тестовых запросов
  - Модель — часть медиатора, писателям wrapper-ов не надо ее делать;
  - Увы, не все компоненты можно описать такими моделями;

- Правила + модель: точно, но трудно — каждому правилу сопоставляем формулу;

- Наблюдение + обучение: неточна, но можно делать адаптацию.

Ссылки I


Distributed DBMS. Sameh Elnikety. Encyclopedia of Database Systems. Ling Liu and M. Tamer Özsu (eds), p. 896–899. Springer US, 2009. http://dx.doi.org/10.1007/978-0-387-39940-9_654


Distributed Database Systems. Kian-Lee Tan. Encyclopedia of Database Systems. Ling Liu and M. Tamer Özsu (eds), p. 894–896. Springer US, 2009. http://dx.doi.org/10.1007/978-0-387-39940-9_701


Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 3rd ed. Prentice-Hall, 2011.


Donald Kossmann. 2000. The state of the art in distributed query processing. ACM Comput. Surv. 32, 4 (December 2000), 422–469. DOI=http://dx.doi.org/10.1145/371578.371598


David Taniar, Clement H. C. Leung, Wenny Rahayu, and Sushant Goel. 2008. High Performance Parallel Database Processing and Grid Databases. Wiley Publishing.