Обработка и исполнение запросов в СУБД (Лекция 5)

Классические системы: принципы построения распределенных СУБД

v5

Георгий Чернышев

Высшая Школа Экономики

chernishev@gmail.com

30 сентября 2020 г.

РСУБД: что нового по сравнению с централизованной?

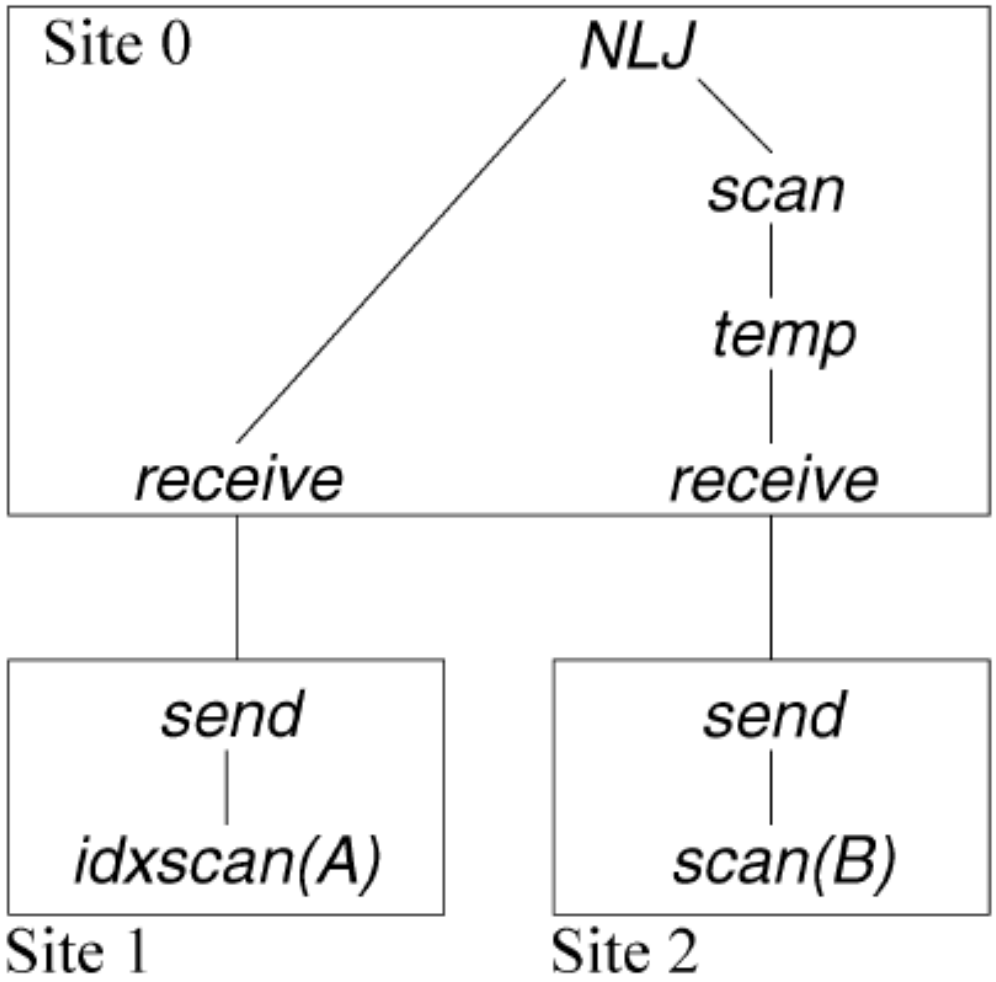


Fig. 2. Example query evaluation plan. [1]

[1] Изображение взято из [Kossmann, 2000]

Распределенная оптимизация

- Отношения могут быть реплицированы и копии могут быть размещены на разных узлах;
- Выбирать AM можно с помощью алгоритма динамического программирования (см. лекцию 2), но теперь не просто $scan(A)$, а $scan(A, S_1)$ и $scan(A, S_2)$. Теперь нельзя сразу же отбрасывать по стоимости — результаты получаются на разных узлах!
- Соединения также производят результаты на разных узлах $\longrightarrow$ концепция interesting sites, аналог interesting orders.
- Выигрыш от кустистых планов выше в распределенных системах.

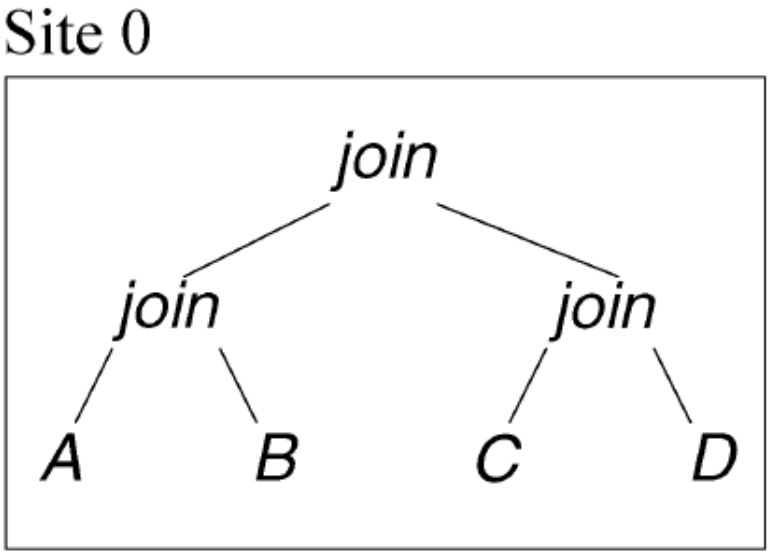
$\longrightarrow$ оптимизация сложнее нежели в централизованных системах

Стоимостные модели для распределенной оптимизации

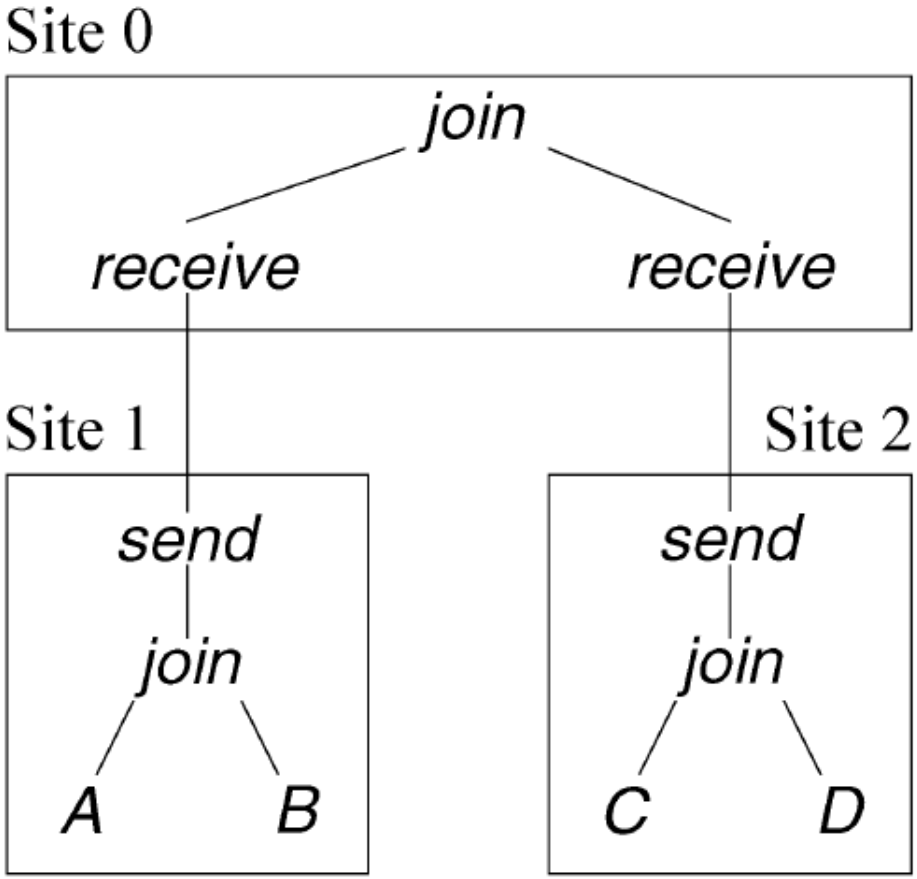
- Модель потребления ресурсов:
  - Централизованная система:
    CPU + disk I/O;
  - Распределенная:
    CPU + disk I/O + network I/O + packing/unpacking + ...;
  - Линейная комбинация с весами;
- Модель времени ответа:
  - Иногда нужен быстрый ответ $\longrightarrow$ нужен внутризапросный параллелизм;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

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

Две модели: пример



Minimum Resource Consumption



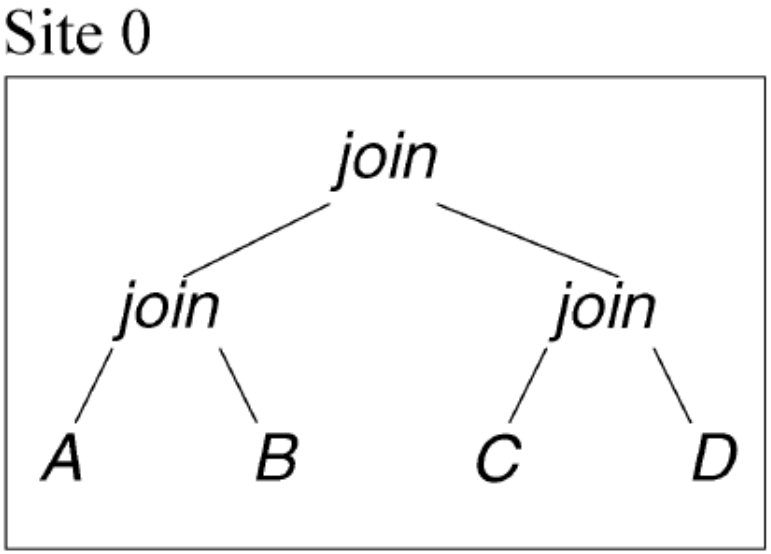
Minimum Response Time

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

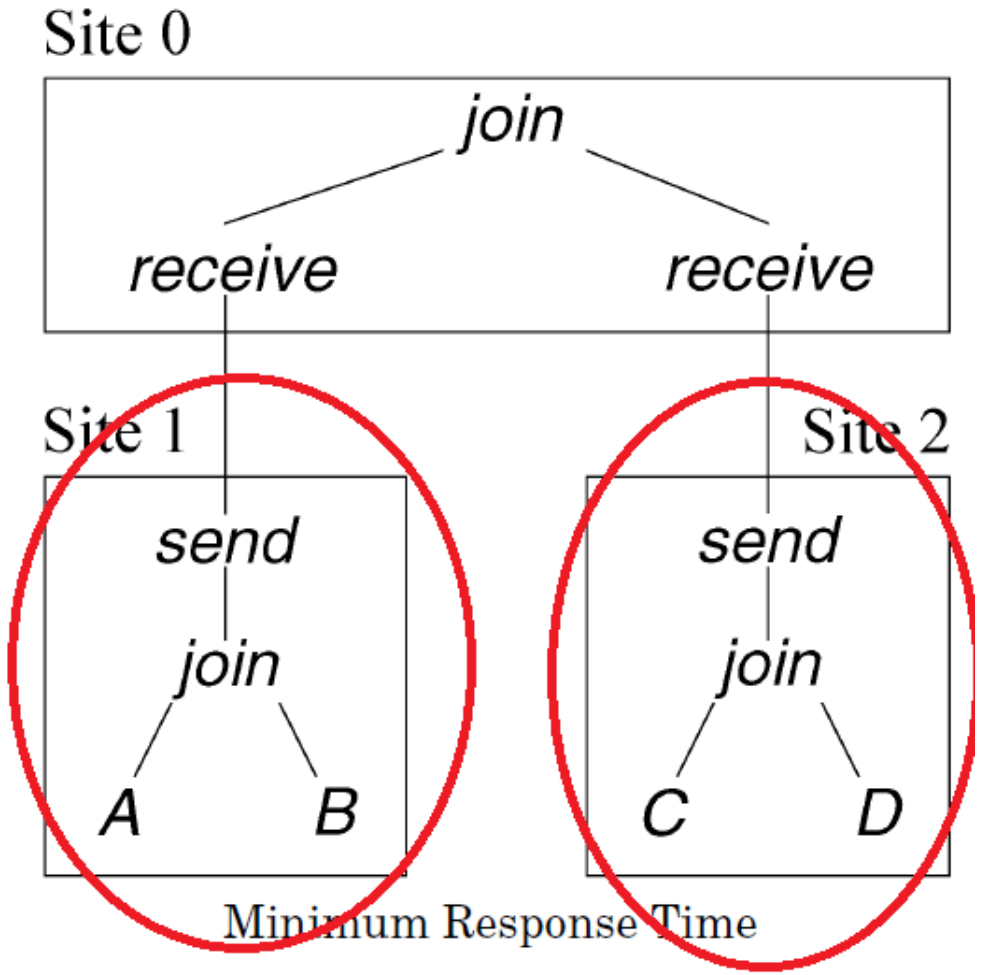
2

[2]

Две модели: независимый внутризапросный параллелизм



Minimum Resource Consumption



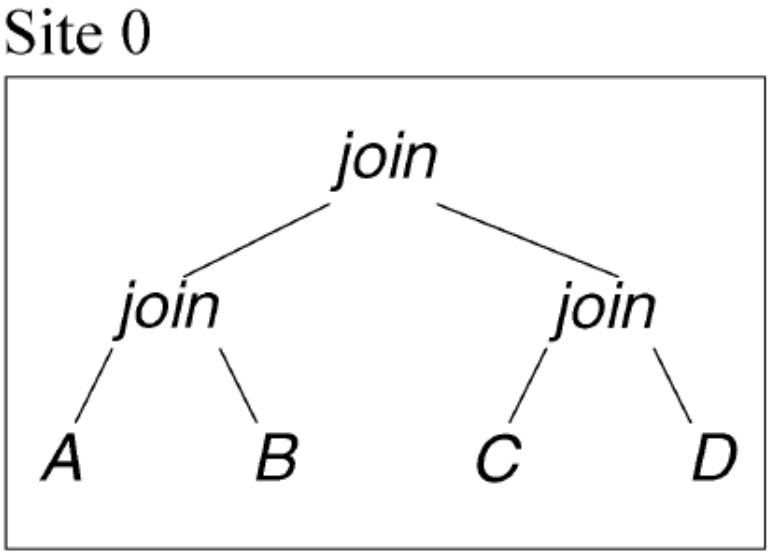
Minimum Response Time

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

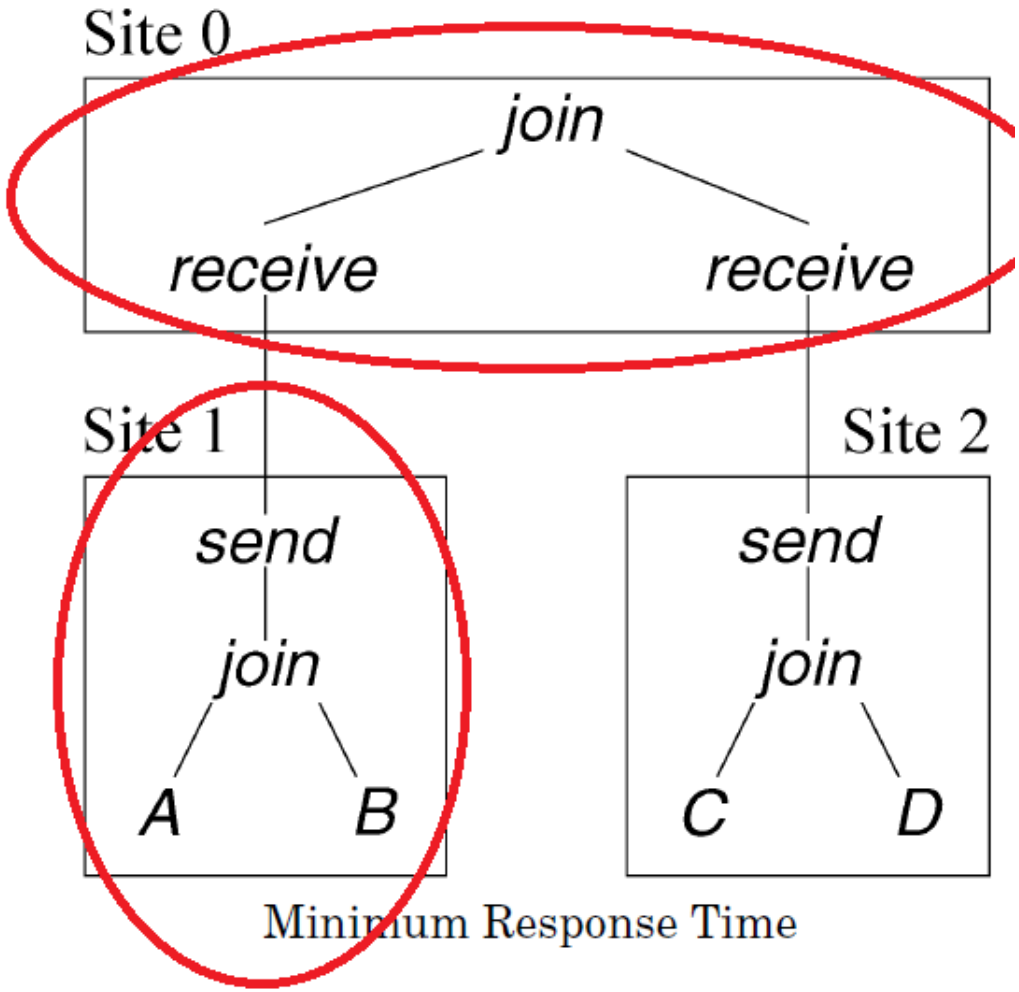
3

[3]

Две модели: pipelining параллелизм



Minimum Resource Consumption



Minimum Response Time

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

4

4

“Наивная” модель времени ответа

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 суммарный объем переданных данных (который передали все операторы).
3. Время ответа группы операторов вычисляется как максимум по потреблению ресурсов отдельных операторов (каждого) и всех разделяемых ресурсов.

Пример

Предположим:

1. На каждом узле по одному процессору, с одним ядром;
2. Все соединения во всех планах работают параллельно (поточно или нет);
3. Каждый оператор соединения стоит 200 секунд работы CPU и нет disk I/O;
4. У сети нет задержек, доставка $A \bowtie B$ и $C \bowtie D$ стоят 130 секунд работы сети (каждая);
5. Посылка и получение данных в/из сети ничего не стоит;
6. Чтение всех таблиц с диска ничего не стоит;

Пример, продолжение

Тогда:

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

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:

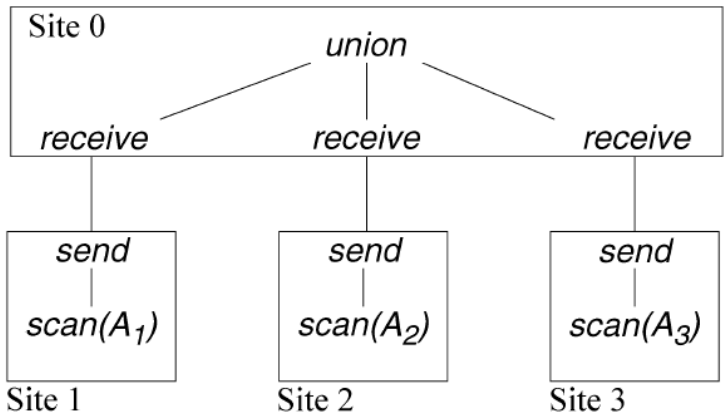


Fig. 5. Example union plan.

5

Как исполнять запросы? II

- Неуместное использование многопоточности:

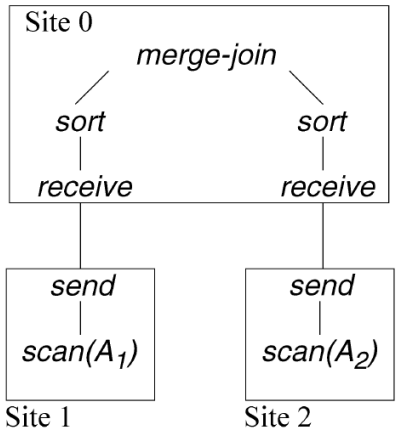


Fig. 6. Example join plan.

[6]

Если не повезет, потоки будут драться за НМЖД (disk thrashing).

- 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: $((A_1 \cup A_2) \bowtie B) \cup (A_3 \bowtie B)$

[5] Изображение взято из [Kossmann, 2000]

[6] Изображение взято из [Kossmann, 2000]

Как исполнять запросы? III

- Попарно не пересекающиеся фрагменты отношений:

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

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

  Чтобы это работало нужно правильно фрагментировать, иначе — каждый с каждым.

- 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

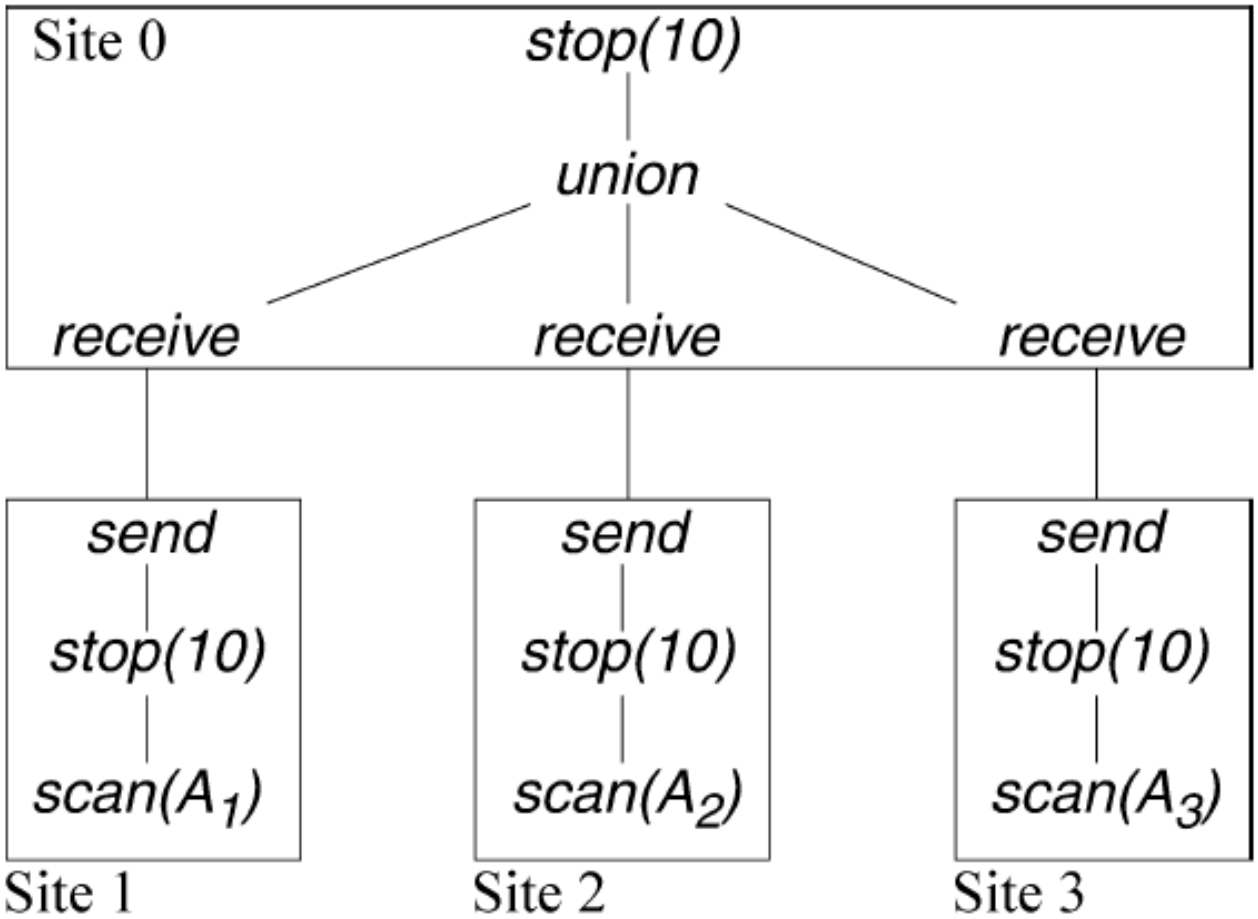


Fig. 7. Example plan for a top N query.

7

[7] 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 (voice, score) = score(voice) + score(looks) ORDERBY ASCENDING

  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 $f(s_1(a), s_2(a)) < f(s_1(b), s_2(b))$.

Как исполнять запросы? VI

Пример (TOP 2):

looks	score
a 1	e 2
b 2	f 3
c 3	**a 8**
d 3	**b 9**
e 3.5	c 10
f 3.8	d 11
...	...

$a = 1 + 8 = 9$

$b = 2 + 9 = 11$

$c = 3 + 10 = 13$

$d = 3 + 11 = 14$

$e = 3.5 + 2 = 5.5$

$f = 3.8 + 3 = 6.8$

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

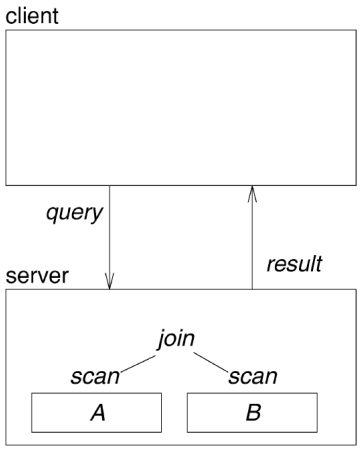


Fig. 8. Query shipping.

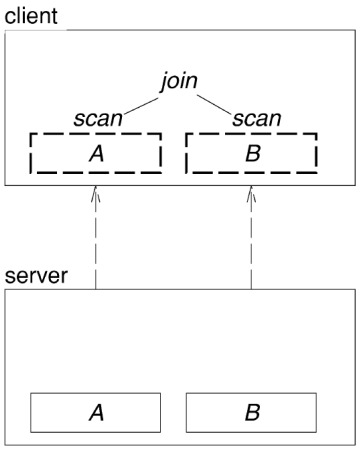


Fig. 9. Data shipping.

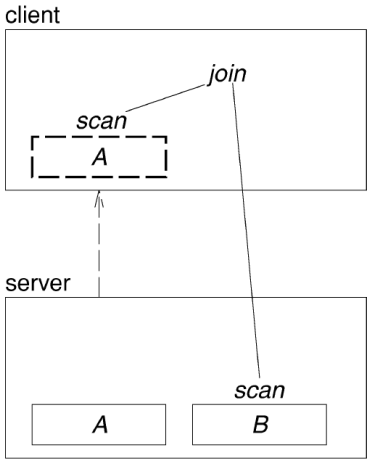


Fig. 10. Hybrid shipping.

8

Стратегии:

- Миграция данных (data shipping);
- Миграция запроса (query shipping);
- Гибридная (hybrid shipping).

[8] 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

Когда и где оптимизировать? Мнений много, некоторые мысли:

- Разбор и перезапись запроса лучше делать на клиенте: разгружаем сервер. Оптимизация — на сервере: понятно если он один, иначе выбор через эвристику.
- Оптимизация на сервере в случае когда их несколько: спрашивать состояние vs угадывать состояние.
- Подходы к оптимизации запросов: сохраненные планы запросов, набор альтернативных планов, реоптимизация, ...
- Двухшаговая оптимизация — самый популярный подход:
  1. абстрактный план: access methods, порядок соединений;
  2. прямо перед выполнением трансформировать план и выбирать узлы.

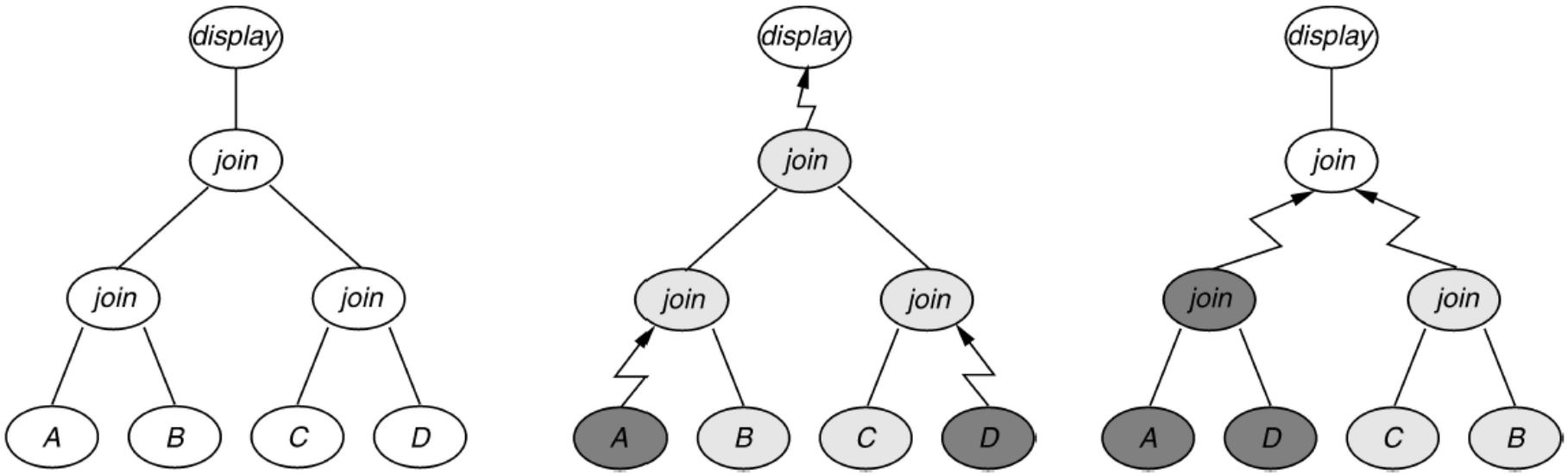
Свойства двухшагового подхода

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.

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


Laura M. Haas, Donald Kossmann, Edward L. Wimmers, and Jun Yang. 1997. Optimizing Queries Across Diverse Data Sources. In Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB '97). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 276-285.

Ссылки II


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