第5章

分布式查询策略的优化

所谓优化，就是在许多可供选择的方法中，选取一种查询代价最低的方法。因为实现优化的技术一般不能得到最优结果，而只是找到一种“良好”策略，所以对各种最优性断言必须详细研究。本章主要介绍基于半连接算法的查询优化、基于直接连接的查询优化算法、R* 中的查询优化算法和 SSD-1 算法。

查询处理策略选择涉及的问题

5.1 查询处理策略选择涉及的问题

查询处理策略的优劣与查询所用数据的存储地点、运算的执行顺序及各种运算的执行方法等密切相关。因此，查询处理策略选择涉及的问题包括如下 3 个方面。

1. 确定查询所需片段的物理副本

一般来说，在执行同样的查询时，对每一个片段尽可能地选取相同的物理副本，而对涉及同一片段的不同子查询则可以在其不同的物理副本上执行。在查询优化时，对于物理副本的选择通常采用以下几种启发式规则。

（1）本场地上的物理副本优先。由于在分布式查询处理中，数据通信代价是影响执行代价的重要因素，因此选择本场地上的物理副本可以减少通信代价。

（2）如果二元运算存在，则尽可能地在本场地上执行。这一规则的目的同样是减少通信代价，因为在一般情况下，执行连接后结果集合的大小要小于两个连接关系的大小。

（3）数据量最小的物理关系应优先选中。

（4）网络通信代价小的物理副本应优先选中。在选择物理副本时不但要考虑副本的大小，而且要考虑网络带宽对通信代价的影响，因为通信代价的计算涉及传输的数据量和两场地之间的网络带宽。

2. 选定运算的执行顺序

选定运算的执行顺序，也就是要确定一个较优的连接、半连接和并运算序列，至于其他运算的执行顺序是不难确定的。值得注意的是，在查询变换后所

产生的查询树中蕴含地定义运算的次序，即按照从树叶到树根的顺序进行运算。然而这并不完全确定优化问题的解法，因为还要求指出在树的同一层上所执行的子表达式的求值顺序。此外，从树叶开始逐步往上运算也不一定就是最好的执行顺序。

3. 选择执行每个操作符的办法

为每个操作符指定合适的物理查询计划，即场地上数据库存取方法的选择，是减少查询执行代价的重要步骤。如尽可能地将同一场地上对同一副本的全部操作在一次数据库访问后一起执行。例如，对于一个关系的选择和投影操作可以同时进行。一般来说，操作符执行方法的选择可以采用集中式数据库中的方法。

在查询优化中，以上 3 个方面彼此间不是互相独立的，而是互相影响的。如操作符的执行顺序会影响中间结果关系的大小，而参与操作符运算的关系的大小会影响执行操作符的算法。同样，物理副本的选择会影响操作符的执行顺序，因此单独考虑某一方面会导致无法获得较好的执行策略。在具体优化时，通常以操作符的执行顺序作为优化的重点，同时考虑其他两个方面的内容。

5.2　基于半连接算法的查询优化

基于连接算法的查询优化

无论是集中式数据库系统还是分布式数据库系统，选择、投影和连接是最常用的 3 种操作。在集中式数据库和分布式数据库中选择和投影操作没有什么不同，可以在局部站点执行。考虑到分布式数据库系统中站点的物理分散性以及关系的分片特性，其上的连接操作很复杂。当连接操作关联的两个关系对象位于不同的站点上时，完成连接操作不可避免地要在站点间进行数据传输，为了降低通信代价很直观的想法是避免网络上不必要的元组的传输，半连接算法正是基于减少数据传输量的思想而做优化的。

5.2.1　半连接操作的定义

半连接(semi-join)是对全连接结果属性列的一种缩减操作，它由投影和连接操作导出，投影操作实现连接属性基数的缩减，连接操作实现左连接关系元组数的缩减。

定义 5.1：假设关系 R 和 S 拥有相同的属性 a，则关系 R 和 S 的半连接为 $\Pi_R(R\infty S)=R\infty\Pi_a(S)$，记为 $R\propto S$，即 $R\propto S=\Pi_R(R\infty S)=R\infty\Pi_a(S)$；关系 S 和 R 的半连接为 $\Pi_S(S\infty R)=S\infty\Pi_a(R)$，记为 $S\propto R$，即 $S\propto R=\Pi_S(S\infty R)=S\infty\Pi_a(R)$。

$R\propto S$ 或者 $S\propto R$ 都是关系 R、S 的半连接运算描述，$R\propto S=\Pi_R(R\infty S)=R\infty\Pi_a(S)$ 保留的是 R 和 S 自然连接结果中关系 R 的属性列，$S\propto R=\Pi_S(S\infty R)=S\infty\Pi_a(R)$ 保留的是 S 的属性列，所以半连接操作也可以理解为利用另一个关系缩减自身关系的元组数，且半连接操作不满足交换律，即 $R\propto S\neq S\propto R$。在执行半连接操作时，会利用数据库系统的站点信息和分片统计信息选择 $R\propto S$ 或者 $S\propto R$。

5.2.2 半连接操作过程和代价估算

关系 R 和关系 S 的连接可用半连接实现，即 $R\infty S=(R\infty\Pi_a(S))\infty S=(R\propto S)\infty S$，其执行示意图如图 5-1 所示。

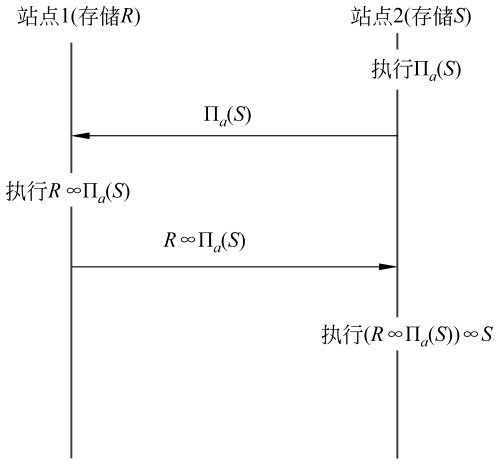

图 5-1 执行示意图

下面讨论这个半连接程序的操作过程和传输代价。其传输代价用 $T=C_0+C_1\times X$ 估算。

(1) 在站点 2 计算关系 S 在属性 a 上的投影 $\Pi_a(S)$。

(2) 把 $\Pi_a(S)$ 的结果从站点 2 传到站点 1，其传输代价为 $C_0+C_1\times\text{Length}(a)\times\text{val}(S,a)$，其中 Length($a$)表示属性 a 的长度，val(S,a)表示关系 S 中属性 a 的个数。

(3) 在站点 1 计算半连接，设其结果为 R'，则 $R'=R\propto S$。实际上，这个操作是执行 $R\infty\Pi_a(S)$。

(4) 把 R'从站点 1 传到站点 2，其传输代价为 $C_0+C_1\times\text{Length}(R)\times\text{card}(R')$，其中：Length($R$)是 R 中元组的长度，card(R')是 R'的元组数。

(5) 在站点 2 执行连接操作 $(R\infty\Pi_a(S))\infty S$。

显然，步骤(1)、(3)、(5)无需传输费用，所以执行这样一个半连接程序，总的传输代价如下：

$$\begin{aligned}T_{\text{semi-jion}}&=C_0+C_1\times\text{Length}(a)\times\text{val}(S,a)+C_0+C_1\times\text{Length}(R)\times\text{card}(R')\\&=2\times C_0+C_1(\text{Length}(a)\times\text{val}(S,a)+\text{Length}(R)\times\text{card}(R'))\end{aligned}$$

由于半连接运算不具有对称性，即没有交换性。因此，另一个等价的半连接程序 $(S\propto R)\infty R$，可能具有不同的传输代价。通过对它们的代价进行比较，就可以确定 R 和 S 的最优半连接程序。

从半连接的执行过程可以看出，相比于全连接来说，半连接比全连接多了一次数据传输过程，但大多数情况下半连接传输的数据总量远比全连接要少得多，即满足 $T_{\text{semi-jion}}<T_{\text{join}}$，此时使用半连接算法能够降低通信代价。半连接操作适用的情况是关系 R 中只有少量元组参与和关系 S 的连接，这时可用半连接算法缩减关系 R 的元组数目。当连接关系较

多时，半连接的种类有很多种，此时需要计算所有的半连接形式的通信代价，从中选出代价最少的半连接方案，并选出传输数据代价最小的站点作为执行连接操作的站点。

5.2.3　基于半连接算法的查询优化案例

假设在站点 1 存储教师基本信息表 Teacher(Tno，Tname，Sex，Major)，其中 Tno 为教师工号(8B)，Tname 为教师姓名(20B)，Sex 为性别(2B)，Major 为专业(20B)。教师基本信息表 Teacher 有 5000 个元组。在站点 2 存储学院基本信息表 College(Cno，Cname，Tno)，其中 Cno 为学院代码(8B)，Cname 为学院名称(20B)，Tno 为学院院长教工号(8B)。学院基本信息表有 20 个元组。现在考虑用户在站点 2 上有一个查询，检索每个学院的名称和学院院长的姓名，其 SQL 语句如下：

```
SELECT Cname, Tname FORM Teacher, College WHERE Teacher. Tno=College. Tno
```

其关系代数表达式如下：

$$\Pi_{Cname,\ Tname}(Teacher \infty College)$$

优化后的查询树如图 5-2 所示。

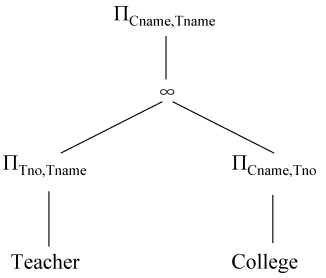

图 5-2　案例查询树

假设每个学院都有一个院长，那么查询结果将包含 20 条记录，并且每条记录为 40B。用半连接方法的步骤如下。

(1) 在站点 2，把 College 关系中的 Tno 值传输到站点 1，即传输关系 Π_{Tno}(College)，它的大小为 8B×20＝160B。

(2) 在站点 1，对被传输过来的 Π_{Tno}(College)和 $\Pi_{Tno,Tname}$(Teacher)关系做连接，然后把要求的属性值从连接结果传输到站点 2 上。也就是传输 $\Pi_{Tno,Tname}$(Teacher)$\infty\Pi_{Tno}$(College))，它的大小为 20B * 28＝560B。

(3) 在站点 2，通过被传输来的 $\Pi_{Tno,Tname}$(Teacher)∞ Π_{Tno}(Colleger))和关系 $\Pi_{Cname,Tno}$(College)做连接来执行查询，然后在站点 2 上将结果呈现给用户。

这个半连接方法中的传输量为 160B＋560B＝720B。在第(2)步中限制 Teacher 的属性和元组传输到站点 2，只传输那些在第(3)步中实际要与 Colleger 元组做连接的属性和元组。此时 Teacher 关系的 5000 个元组中只有 20 个元组传过去。

如果不采用半连接程序法，而直接采用连接法，例如在站点 2 执行连接操作，那么需要把关系 $\Pi_{Tno,Tname}$(Teacher)从站点 1 传到站点 2，所需的代价为 28B×5000＝140 000B，这个代价太大了。

5.3　基于直接连接的查询优化算法

在广域网中由于数据传输速率的限制，执行查询操作所花费的通信代价时间要比 I/O 代价大得多，这种情况下查询所花的代价可以用通信代价来衡量而忽视 I/O 代价。在高速局域网中或专线网络中，数据的传输速率很快，传输数据的通信代价和 I/O 代价

所占的比重差不多或者前者消耗的时间更少,此时系统综合考虑通信代价和 I/O 代价。高速局域网和专线网络中高速的数据传输特性使得此种网络环境更加注重查询响应时间,而不是通信代价,所以在局域网或专线网络中执行连接操作时总是从本地站点传输整个关系到另一个站点,对此所做的优化称为直接连接查询优化。

5.3.1 直接连接操作的策略

直接连接操作依据参与连接的两个关系是否在同一站点而采取不同的操作策略。当两个关系在同一站点时,与集中式数据库一样可采用嵌套循环连接算法和基于排序的连接算法。

(1) 嵌套循环连接算法。顺序扫描外层关系 R,并对 R 的每个元组扫描内层关系 S,查找在连接属性上相等的元组并将其组合起来形成结果的一部分,这种方法需要扫描一次关系 R 和 Card(R)次关系 S 才能找到匹配的元组。

(2) 基于排序的连接算法。基于连接属性对两个关系先后进行排序和扫描操作,使得到的结果中包含以上两种操作所获得的匹配元组的组合。此方法虽然扫描的次数不多,但是增多了排序的代价。

对于不同站点上的关系 R 和 S 的连接,除考虑局部代价外还需要考虑传输代价。影响传输代价的因素有两个方面:传输方式和连接站点。

传输方式有两种:整体传输方式和按需传输方式。

(1) 整体传输方式:假设 $R\infty S$ 这一连接操作,外层关系为 R,内层关系为 S。若对关系 S 进行传输,则将它进行保存;若对关系 R 进行传输,则陆续来到的 R 元组可以被关系 S 直接使用,且关系 R 不保存。

(2) 按需传输方式:在一次只传输一个元组的前提下,针对那些需要执行连接操作的元组进行传输,不需要临时存储,该传输方式的传输代价很高,因为交换信息的次数过于频繁,每次提取都需要进行一次信息交换。

选择连接站点的方法有以下三类:

(1) 将 R 所在站点作为连接站点。

(2) 将 S 所在站点作为连接站点。

(3) 使用其他站点作为连接站点。

5.3.2 嵌套循环连接算法

嵌套循环连接算法是一种最简单的连接算法,其原理是对连接操作的两个关系对象中的一个仅读取其元组一次,而对另一个关系对象中的元组将重复读取。嵌套循环连接算法的特点是可以用于任何大小的关系间的连接操作,不必受连接操作所分配的内存空间大小的限制。对于嵌套循环连接算法,可根据每次操作的对象大小分为基于元组的嵌套循环连接和基于块(block)的嵌套循环连接。

假设有关系 $R(A, B)$和关系 $S(B, C)$,分别有 Card(R) $=n$ 和 Card(S)$=m$,现在要执行两个关系在属性 B 上的连接操作,操作过程如图 5-3 所示。

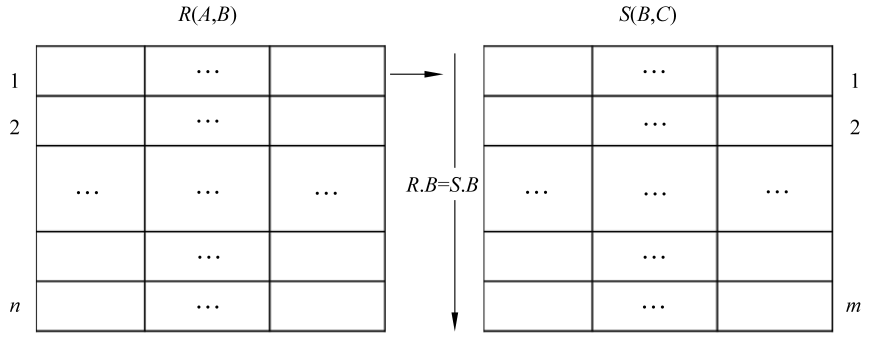

图 5-3　嵌套循环连接算法操作过程

1. 基于元组的嵌套循环连接算法

基于元组的嵌套循环连接是最简单的形式，其中循环以关系中的元组为单位进行操作，假设有关系 $R(A, B)$和关系 $S(B, C)$，则 R 和 S 基于元组的嵌套循环连接具体的执行算法如下：

```
Result=ϕ
For each tuple r in R
      For each tuple s in S
          If r.B=s.B Then
            join r and s as tuple t
            output t into Result
          End If
      End For
End For
Return Result
```

其中，对循环外层的关系通常称为外关系，而对循环内层的关系称为内关系。在执行循环嵌套连接时，仅仅对外关系进行 1 次读取操作，而对内关系则需要进行反复读取操作。如果不进行优化的话，这种基于元组的执行代价很大，磁盘读取代价计算最多可能多达 Card(R)×Card(S)。因此，通常对这种算法进行修改，以减少嵌套循环连接的磁盘读取代价。基于元组的嵌套循环连接算法优化通常使用两种方法：一种方法是使用连接属性上的索引，以减小参与连接元组的数量；另一种方法是通过尽可能多地使用内存以减少磁盘读取数量。

2. 基于块的嵌套循环连接算法

基于块的嵌套循环连接算法是通过尽可能多地使用内存，减少读取元组的 I/O 次数，其中，对连接操作的两个关系的访问均按块（也称为页面）进行组织，同时使用尽可能多的内存来存储嵌套循环中外关系的块。

与基于元组的方法相似，将连接操作中的一个对象作为外关系，每次读取部分元组到

内存中,整个关系只读取一次;而另一个对象作为内关系,反复读取到内存中执行连接。对于每个逻辑操作符,数据库系统都会分配一个有限的内存缓冲区。假设为连接操作分配的内存缓冲区大小为 M 块,同时有 Block(R)≥Block(S)≥M,即连接的两个关系都不能完全读取到内存中。为此,首先选取较小的关系作为外关系,这里选择关系 S,将 1 到 $M-1$ 块分配给关系 S,而第 M 块分配给关系 R,将外关系 S 按照 $M-1$ 块的大小分为多个子表,并重复地将这些子表读取到内存缓冲区中,用于重复地依次读取关系 R 的每一个块。对于内存缓冲区中元组的连接操作,先在 $M-1$ 个块的外关系 S 元组的连接属性上构建查找结构,再从内关系 R 在内存中的块中取元组,通过查找结构与 S 中的元组连接。基于块的嵌套循环连接方法的原理如图 5-4 所示。

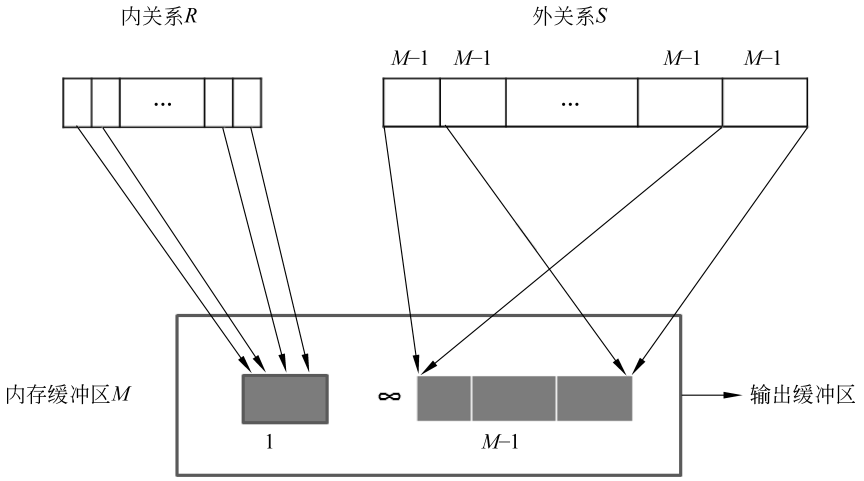

图 5-4 基于块的嵌套循环连接方法的原理

假设有关系 $R(A, B)$和关系 $S(B, C)$,S 是内关系,R 是外关系,则 R 和 S 基于块的嵌套循环连接具体的执行算法如下:

```
Result=ϕ
Buffer=M
For each M-1 Block in Block(S)
  Read M-1 Block of Block(S) into Buffer
      For each Block in Block(R)
          Read 1 Block of Block(R) into Buffer
          Join M-1 Block of Block(S) and 1 Block in Block(R)s in Buffer
          output t into Result
      End For
End For
Return Result
```

3. 嵌套循环连接算法的代价估计

对于两个关系 R 和 S,如果使用基于元组的嵌套循环连接方法,则需要对每个元组

的读取产生 1 次磁盘 I/O。因此，假设两个关系的元组数量分别为 Card(R)和 Card(S)，则基于元组的嵌套连接方法的执行代价为 Card(R)×Card(S)，即两个关系大小的乘积。

对于基于块的嵌套循环连接算法来说，假设两个连接关系 R 和 S 占用的块分别为 Block(R)和 Block(S)，M 为内存缓冲区大小。在嵌套循环过程中使用 S 作为外关系，每一次迭代时首先读取 $M-1$ 块 S 的内容到内存缓冲区，再每次读取 R 的 Block(R)中的 1 块的内容到内存中与 $M-1$ 块的 S 内容执行连接。连接的代价用以下公式计算：

$$C_{\text{join}} = \underset{(M-1)}{\text{Block}(S)} \times (M-1+\text{Block}(R))$$
$$= \text{Block}(S) + \underset{(M-1)}{\text{Block}(S)} \times \text{Block}(R)$$

从以上公式可以看出，选择较小的关系作为连接的外关系可以获得较小的执行代价，因此通常选择较小的关系作为外关系。如果连接关系的 Block(R)、Block(S)的值很大，且远远大于内存缓冲区大小 M 时，可以认为连接的代价近似等于 Block(R)×Block(S)。虽然嵌套循环连接的执行代价看上去较高，但是这种算法能够适用于任意大小的关系之间的连接执行。因此，嵌套循环连接算法依然广泛应用于现有的数据库系统中。

5.3.3　基于排序的连接算法

基于排序的连接算法(sort-based join algorithm)是直接连接算法中的另外一种常用方法，其首先将两个关系按照连接属性进行排序，然后按照连接属性的顺序扫描两个关系，同时对两个关系中的元组执行连接操作。因为数据库中关系的大小往往大于连接操作可用内存缓冲区的大小，所以对关系的排序通常采用外存排序算法，即归并排序算法。

1. 归并排序算法

简单的归并排序算法的执行可以划分为两个阶段。

(1) 对关系进行分段排序，即首先将需要排序的关系 R 划分为大小为 M 块的子表，其中 M 是可用于排序的内存空间的数量，以块为单位，再将每个子表放入内存中采用快速排序等主存排序算法执行排序操作，这样可以获得一组内部已排序的子表。对关系进行分段排序过程如图 5-5 所示。

(2) 对关系的子表执行归并操作，即按照顺序从每个排序的子表中读取一块的内容放入内存中，在内存中统一对这些块中的记录执行归并操作，每次选择最小(最大)的记录放入输出缓冲区中，同时删除子表中相应的记录。当子表在内存中的块被取空时，从子表中顺序读取一新块放入内存中继续执行归并操作。归并操作的过程如图 5-6 所示。

归并操作过程同时对多个子表执行归并操作，因此也称为两阶段多路归并排序。需要说明的是，第二阶段的归并操作执行的条件是关系的子表数量小于排序操作可用内存的块数 M，这样才能保证同时对所有子表进行归并操作。因此，两阶段归并执行的条件是关系的大小 Block(R)$\leqslant M^2$。如果关系的大小 Block(R)$>M^2$，则需要嵌套执行归并排序算法，使用三阶段或更多次的归并操作。

2. 基于排序的连接算法

基于排序的连接算法主要是对已经按照连接属性排序的两个关系，按照顺序读取关

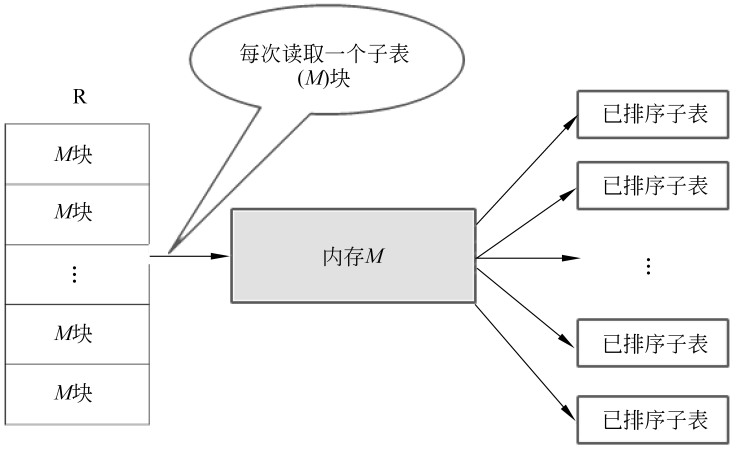

图 5-5 分段排序过程

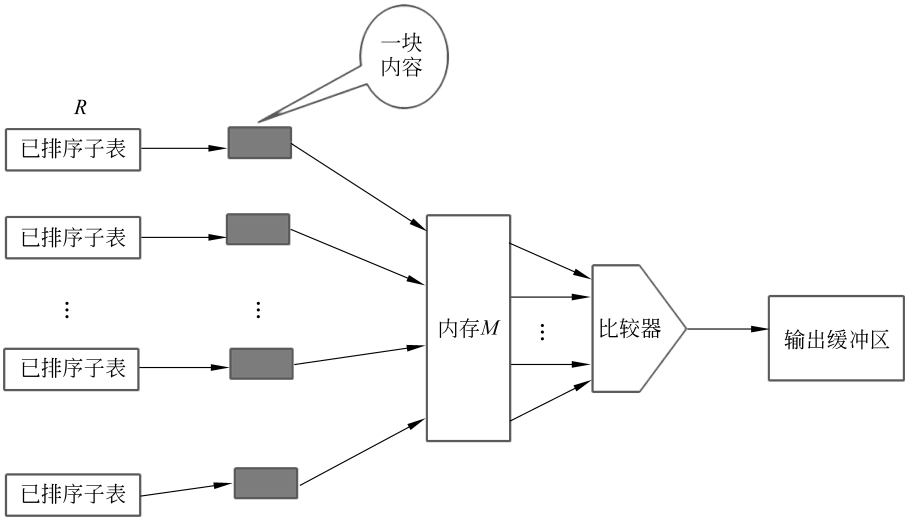

图 5-6 归并操作过程

系中的块到内存中执行连接操作。基于排序的连接算法执行过程如图 5-7 所示,其中先使用内存对关系 R 和 S 进行排序,再基于归并方法按顺序依次连接关系中的元组。

3. 基于排序的连接算法代价估计

在基于排序的连接算法中,假设在排序阶段使用的是两阶段多路归并排序,关系的大小满足条件 $\text{Block}(R)\leqslant M^2$ 和 $\text{Block}(S)\leqslant M^2$。

在算法排序阶段的执行代价包括如下。

(1) 对关系的子表执行排序所需的一次读(读子表数据)和一次写(子表排序结果写入磁盘)的代价,为 $2(\text{Block}(R)+\text{Block}(S))$。

(2) 多路归并时的读写代价为 $2(\text{Block}(R)+\text{Block}(S))$。

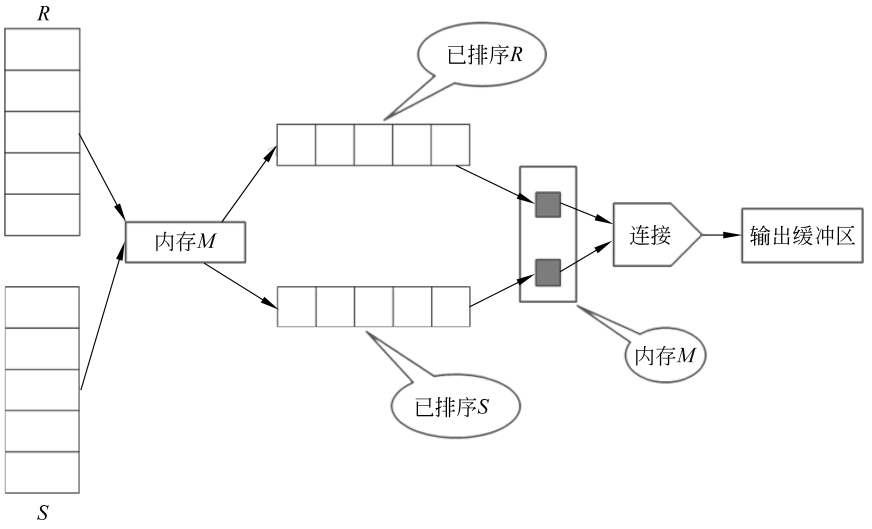

图 5-7　基于排序的连接算法执行过程

算法排序阶段的执行代价为 4(Block(*R*)＋Block(S))。

在归并连接阶段需要对关系执行一次读操作，因此，基于排序的连接算法的执行代价为 5(Block(*R*)＋Block(S))。

5.3.4　站点依赖算法

在分布式数据库系统中，关系或者关系的某个片段总是分布在不同的站点上，当两个关系做连接操作时，如果在数据传输量最小甚至无数据传输方式下得到正确的结果，此时可获得最佳的性能。

假定两个关系 R_1 和 R_2 的水平分片分别存放在站点 S_1 和 S_2 上，数据分布的初始状态如表 5-1 所示。

表 5-1　数据分布的初始状态

关系 \ 场地		站点 S_1	站点 S_2
关系	R_1	R_{11}	R_{12}
	R_2	R_{21}	R_{22}

对关系 R_1 和 R_2 在连接属性 A 上的自然连接 $R_1 \infty R_2$，如果其结果可以通过合并同一站点上两个关系片段的自然连接的结果集得到，即 $R_1 \infty R_2 = (R_{11} \infty R_{21}) \cup (R_{12} \infty R_{22})$，则该策略是一种有效的策略。在该策略中，由于连接操作所涉及的片段总能在本站点找到可关联的元组，因此连接操作可以在站点间不发生数据传输的情况下进行，而且还可以利用本地站点中数据片段的索引信息提升局部处理的性能。上述情况即为“站点依赖”。