

# 数据库系统原理研究报告

—— 分布式数据库系统的查询优化技术



同濟大學  
TONGJI UNIVERSITY

院 系 电子与信息工程学院

专 业 计算机科学与技术

姓 名 庄镇华

学 号 1853790

题 目 分布式数据库系统的查询优化技术

指导老师 李文根

## 目录

一、分布式数据库系统查询的难点.....	1
1.1 分布式数据库系统结构特点.....	1
1.1.1 定义.....	1
1.1.2 结构.....	1
1.2 分布式数据库系统查询难点.....	2
1.2.1 最优解参数更多的问题.....	2
1.2.2 子站点分配与合并问题.....	2
二、分布式查询优化技术.....	3
2.1 查询优化概述.....	3
2.1.1 搜索空间.....	3
2.1.2 搜索策略.....	3
2.1.3 分布式代价模型.....	3
2.2 基于半连接算法的查询优化算法（减少数据传输量）.....	4
半连接的定义.....	4
算法执行过程.....	4
通信代价定性估算.....	5
2.3 基于直接连接的查询优化算法（减少查询响应时间）.....	5
站点依赖算法.....	6
分片和复制算法.....	7
Hash 划分算法.....	8
2.4 基于查询图的查询优化算法.....	9
2.4.1 基于链查询的 CHAIN 算法.....	9
2.4.2 Kruskal 启发式算法.....	10

2.5 基于智能改进的查询优化算法.....	12
2.5.1 基于蚁群算法的查询优化.....	12
2.5.2 基于鱼群算法的查询优化.....	12
2.5.3 基于并行遗传-蚁群算法的查询优化 .....	13
三、个人感悟与见解.....	13
3.1 学习感悟.....	13
3.2 个人见解.....	14
四、参考文献.....	16



## 一、分布式数据库系统查询的难点

### 1.1 分布式数据库系统结构特点

#### 1.1.1 定义

分布式数据库是分布在计算机网络上的多个逻辑上相关的数据库的集合；分布式数据库管理系统（Distributed Database Management System）是管理分布式数据库的软件，它提供了让用户透明访问分布式数据库的访问机制。通俗地讲，就是物理上分散而逻辑上集中的数据库系统。

在分布式数据库系统中，每个连接在计算机网络上的逻辑单位是能够独立完成本地工作的计算机，这些计算机被称为数据处理站点。当某个站点用户的查询操作涉及的数据未全部存储于该站点时，本次查询请求就需要通过分布式查询优化机制将本次的查询请求操作分发到不同的站点上执行，之后将各个站点查询到的结果合并传到接收查询请求的站点。

#### 1.1.2 结构

分布式数据库系统的运行依赖于网络环境，对分布、异构和自治的数据进行全局统一管理，数据分散存储到各个物理场地上是依靠分片技术和副本复制技术实现的。

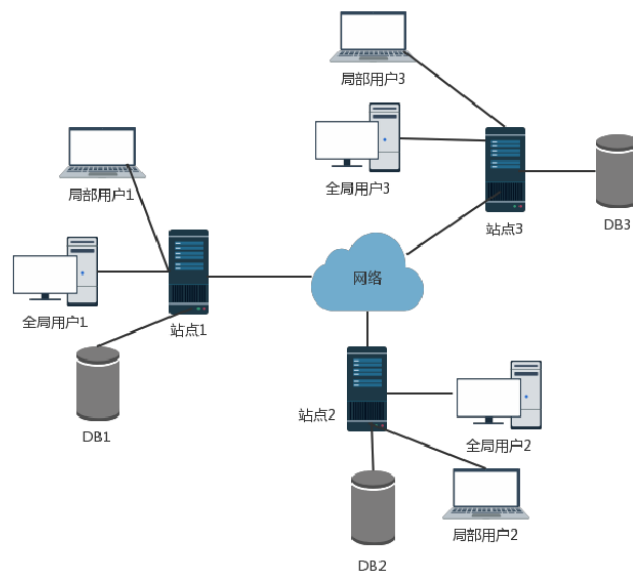


图 1-1 分布式数据库系统物理结构图

**物理结构：**图 1-1 给出了典型的 DDBS 物理结构。位于各个物理位置的计

计算机或服务器及其上部署的数据库系统组成了分布式数据库系统中的一个站点，站点上的数据库和局部用户由其上的计算机或服务器控制，不同场地之间通过网络通信链路进行通信，网络可以是局域网或者广域网。

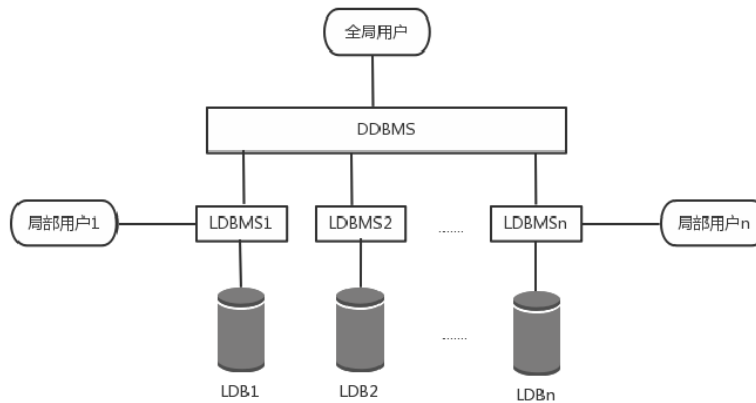


图 1-2 分布式数据库系统简单逻辑结构图

**逻辑结构：**DDBS 简单的逻辑结构图如图 1-2 所示，分布式数据库系统作为一个逻辑统一的整体单元结构，分布式数据库管理系统（DDBMS）对系统中的所有用户和数据进行统一管理；局部数据库管理系统（LDBMS）类似于集中式数据库管理系统，主要对本地站点的数据进行管理。

## 1.2 分布式数据库系统查询难点

**结构决定性质，分布式系统的结构决定了查询的困难。**

### 1.2.1 最优解参数更多的问题

集中式查询处理程序处理给定查询的策略是从所有等效查询中选出一个接近最优的关系代数查询；而分布式系统除了考虑关系代数运算顺序这一问题外，还需要考虑数据在站点间的传输方法以及选择哪些站点处理数据。分布式查询处理的最优解问题的参数量比集中式查询处理大得多，使得分布式查询处理问题变得异常复杂。

### 1.2.2 子站点分配与合并问题

分布式数据库的物理独立性导致了分布式系统中的查询操作往往要对多个站点进行查询处理，其中涉及到了子查询的站点分配、lease holder 的选取、时间戳的设置、多个站点查询结果合并时的路由选择等问题，这使得分布式数据库的查询操作远比集中式数据库复杂的多。

## 二、分布式查询优化技术

### 2.1 查询优化概述

查询优化指的是生成查询执行计划（Query Execution Plan）的过程。为了实现选择最优的执行策略，查询优化的执行主要涉及搜索空间（search space）、搜索策略（search strategy）和查询代价（cost model）模型三个概念。



#### 2.1.1 搜索空间

搜索空间是指将输入的查询语句依据变换规则生成多个操作符执行顺序和执行方法不同的等效的查询执行计划（运算树），这些执行计划的等效性体现在它们获得的查询结果是相同的。一个包含较多关系和操作符种类的复杂查询按照关系连接顺序和操作顺序的转换后生成的等价运算数的数量可能很大。

包含  $N$  个关系的等价查询运算数的数目为  $O(N!)$ ，可以采用启发式规则来缩小搜索空间范围，常见的规则包括：对基础关系的运算先做投影和选择运算，再做连接运算；投影运算应尽可能下移并优先执行；避免查询不需要的笛卡尔积，可将笛卡尔积转换为等效的等值连接等。

#### 2.1.2 搜索策略

搜索策略定义了对搜索空间每个等效查询执行计划代价估计的顺序，有效降低了选择执行计划的代价。目前有两种主要的搜索策略分类法：

-  自顶向下方法：从表达式树的根节点向叶节点遍历，估计每个节点可能的执行方法，计算所有组合的代价并从中选出最优执行计划。
-  自底向上方法：从表达式树的叶节点向上遍历，选出最优的子表达式的执行方法并组合出最优执行计划。常用的搜索策略包括动态规划方法、启发式方法和爬山法等算法。

#### 2.1.3 分布式代价模型

优化程序的代价模型由预测操作代价的函数、数据库信息统计、中间结果大小估计几部分组成，响应时间是衡量一个数据库性能的重要指标，所以代价经常用操作执行的时间来衡量。

所花费的全部时间的计算公式：

$$T_{total\_time} = T_{CPU} \times \#insts + T_{I/O} \times \#I/O_S + T_{MSG} \times \#msgs + T_{TR} \times \#bytes$$

$T_{CPU}$  是执行查询命令式 CPU 的指令执行时间,  $T_{I/O}$  是查询执行时在本地磁盘上读写数据库数据所化的时间, 这两部分都是本地站点相关。公式的后两部分用来计算站点之间传输数据所花费的通信代价。

当响应时间成为优化程序的主要考虑的性能指标时, 需要综合考虑本地处理和并行通信的问题, 通用的响应时间的计算方法如下:

$$T_{response\_time} = T_{CPU} \times seq\_ \#insts + T_{I/O} \times seq\_ \#I/O_S + T_{MSG} \times seq\_ \#msgs + T_{TR} \times seq\_ \#bytes$$

## 2.2 基于半连接算法的查询优化算法 (减少数据传输量)

选择、投影和连接是数据库查询最常用的三种操作。在集中式数据库和分布式数据库中选择和投影操作基本相同。但由于分布式数据库系统中站点的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, 为了完成连接操作不得不在站点间进行数据传输, 而显然避免网络上不必要的元组传输可以降低通信代价。广域网中通信代价在查询代价占比很大, 半连接算法正是基于减少数据传输量的思想进行优化的。

### 半连接的定义

半连接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。假设关系 R 和 S 拥有相同属性 a, 公式(2-1)、(2-2)为半连接的描述。

$$R \bowtie S = \pi_R(R \bowtie S) = R \bowtie \pi_a(S) \quad (2-1)$$

$$S \bowtie R = \pi_S(S \bowtie R) = S \bowtie \pi_a(R) \quad (2-2)$$

公式 (2-1) 保留的是 R 和 S 自然连接结果中关系 R 的属性列, 公式(2-2)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元组数, 并且半连接操作不满足交换律。

### 算法执行过程

半连接操作的执行过程包括了连接关系的投影运算和连接运算，执行过程如表 2-1 所示。

表 2-1 半连接执行过程

步骤	执行操作
(1)	在站点 S2 上对连接属性字段做投影去重操作 $\Pi_a(S)$
(2)	将第一步的结果传送到站点 S1
(3)	在站点 S1 上执行半连接操作 $R \bowtie \Pi_a(S)$ ，缩减传输关系 R 的元组
(4)	将缩减后的关系 R 从站点 S1 传输到站点 S2
(5)	在站点 S2 执行连接操作 $(R \bowtie \Pi_a(S))$

### 通信代价定性估算

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比全连接要少的多，即满足  $T_{\text{semi-join}} \ll T_{\text{join}}$ ，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。

## 2.3 基于直接连接的查询优化算法（减少查询响应时间）

在高速局域网中或专线网络中，数据的传输速度很快，传输数据的通信代价和 I/O 代价所占的比重差不多，此时系统综合考虑通信代价和 I/O 代价，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，对此所做的优化成为直接连接查询优化，包括利用站点依赖或者 Hash 划分算法划分数据减少站点之间的依赖关系而降低查询代价。



如果参与连接操作的两个关系  $R$  和  $S$  存储于不同的站点，那么完成连接操作产生的代价包括通信代价和局部处理代价。此时，系统可以采用一次传输整个连接关系的方式，例如为了执行连接操作  $R \bowtie S$ ，我们可以选择传输  $R$  或者  $S$ ；系统也可以依据查询语句中的判断条件只传输需要连接的元组，且一次只传输一个元组。完成连接操作有三种站点选择策略：选择  $R$  所在站点、 $S$  所在站点或者不包含  $R$  与  $S$  的其它站点。

### 站点依赖算法

站点依赖算法是指如果分布式数据库满足站点依赖条件，那么这种模式下连接操作可以在每个局部站点独立执行，不同站点之间不需要传输数据进而节省传输代价。

**站点依赖条件：**两个片段关系  $R_i$  和  $R_j$  在属性  $A$  上满足  $F_{is} \cap F_{jt} = \emptyset$ ，则称  $R_i$  和  $R_j$  在属性  $A$  上站点依赖，即  $R_i \bowtie_A R_j = \cup (F_{is} \bowtie_A F_{jt})$ 。其中  $s$  和  $t$  代表站点编号， $F_{is}$  表示关系  $R_i$  在站点  $s$  上的片段， $F_{jt}$  表示关系  $R_j$  在站点  $t$  上的分片。

### 推论：

属性集  $B \subseteq A$  且  $R_i$  在属性  $A$  上站点依赖  $\rightarrow R_i$  和  $R_j$  在属性  $B$  上站点依赖。

属性  $A$  由属性  $B$  的值决定，关系  $R_i$  和  $R_j$  关于  $A$  站点依赖  $\rightarrow R_i$  和  $R_j$  关于属性  $B$  也站点依赖。

关系  $R_j$  和  $R_k$  关于属性  $B$  站点依赖， $R_i$  和  $R_j$  关于属性  $A$  站点依赖  $\rightarrow (R_i \bowtie_A R_j \bowtie_B R_k) = \cup (F_{is} \bowtie_A F_{js} \bowtie_B F_{ks})$ 。

### 举例：

		站点	
		S1	S2
关 系	R1	F11	F12
	R2	F21	F22

若满足站点依赖条件，则  $R1 \bowtie R2 = (F11 \bowtie F12) \cup (F12 \bowtie F22)$ 。

**优点：**无数据传送；可进行并行运算；可以利用本地索引提高效率。

### 分片和复制算法

当分布式数据库无法满足站点依赖条件时，可以采用分布式数据库的一些冗余的分配数据算法来解决。即当查询不能在无数据传送方式下处理，可采用分片和复制算法。

**原理：**该算法的思想是保留一个关系在站点间的分片状态，其余关系在站点间通过片段之间的复制传输使得每个站点都能得到其余关系的一个完整副本。之后，利用这些关系的副本和第一个关系的分片数据在各个站点间做本地连接操作，将多个站点间的连接结果合返回查询结果。当涉及两个以上的关系时，分片和复制算法只让一个关系在站点间以分片形式存储，其余关系则在站点间存储完整副本。

表 2-2 分片复制算法执行流程

### 分片复制算法(Q,S,R)

For 每个保持分片状态的关系  $R_i$ ,  $i = 1, 2, \dots, n$

For 每个包含关系  $R_i$  的一个片段的站点  $S_j, j=1, 2, \dots, m$

计算在站点  $S_j$  执行子查询的完成时间  $FT(Q, S_j, R_i)$

计算关系  $R_i$  保持分片状态下的响应时间  $T_i = \max(FT(Q, S_1, R_i), \dots, FT(Q, S_n, R_i))$

选择  $R_k = \min(T_1, T_2, \dots, T_n)$

**举例：**

		站点	
		S1	S2
关 系	R1	F11	F12
	R2	F21	F22

分片复制后：

		站点	
		S1	S2
关 系	R1	F11	F12
	R2	R2	R2

**优点：**可进行并行计算；在一定程度上可利用本地索引；无需满足站点依赖条件。

### Hash 划分算法

当关系间不满足站点依赖时，也可以选用 Hash 划分算法实现站点依赖。

**原理：**Hash 划分算法的思想是根据关系元组中连接属性的 Hash 值将元组发送到 Hash 值对应的站点上，这样不在同一个站点上的元组之间因为值不同所以不能做连接操作，而同一个站点上的不同关系的片段之间因为存在连接属性值相同的元素而可以做连接操作，这正是站点依赖所符合的特性，即通过 Hash 划分算法保证了关系间的站点依赖。

**步骤：**可以通过如下划分方式将关系中的元组划分到对应站点上：

1) 如果系统中站点的分数为  $i$ ，关系上的连接属性的属性值为  $a$ ，则可以用的 Hash 函数可以表示为： $\text{Hash}(a) = (a \bmod i) + 1$ ，即用  $a$  的值除以  $i$  得到的余数加一即为该元组所需要分配的站点位置，关系中的属性通过这个 Hash 函数而被划分到编号为 0 到  $i - 1$  的  $i$  个站点上。

2) 还可以用连接属性的取值范围对一个关系或者片段做 Hash 映射。例如，如果连接属性  $A$  在值域  $[\text{low}, \text{high}]$  上取值，那么该区域可以分隔成  $[\text{low}, a_1]$ ,  $(a_1, b_1]$ ,  $\dots$ ,  $(a_{i-1}, \text{high}]$  这  $i$  个存放在不同站点上的互不相交的区域，使得属性  $A$  的取值落在区间  $[\text{low}, a_1]$  时发往站点  $S_1$ ，在区间  $(a_j, b_j]$  时发往站点  $S_{j+1}$ ， $1 \leq j \leq i-2$ ，以及属性  $A$  的值在区间  $(a_{i-1}, \text{high}]$  时发往站点  $S_i$ ，这被称作值域分区。如果关系在连接属性上的值是分布均匀的，那么值  $a_1, a_2, \dots, a_{i-1}$  应等距分隔；否则，应当对区间的边界点做调整，使得每个区间中有大致相等的元组。

**举例：**以下表为例，假设连接属性  $A$  取整数值，可以对其使用 Hash 函数  $\text{Hash}(a)$ ：若  $a$  的值为偶数，则令  $\text{Hash}(a)=0$ ；如果  $a$  的值为奇数，则令

		站点	
		S1	S2
关 系	R1	F11	F12
	R2	F21	F22

Hash(a)=1。如果一个元组的 Hash 值为 1 就把该元组分配到 S1 站点，如果一个元组的 Hash 为 0 就将该元组发送到 S2 站点。经过 Hash 划分后，原来的分片片段 F22 被分为两个子段  $F22O$  和  $F22E$ ，其中 O 和 E 分别代表连接属性取值为奇数和偶数。同样 F21 被分为两个子段  $F21O$  和  $F21E$ ， $F21O$  和  $F22O$  在站点 S1 上合并成一个新片段  $F21'$ ， $F21E$  和  $F22E$  在站点 S2 上合并成一个新片段  $F22'$ 。显然  $\Pi_A(F11') \cap \Pi_A(F22') = \Phi$ ，这说明 R1 和 R2 在新组成的片段分布情况下在属性 A 上站点依赖。这种情况下  $R1 \bowtie R2$  可以由  $(F11' \bowtie F21') \bowtie (F12' \bowtie F22')$  得到。如果属性 A 上的取值不是整数，那么可以通过设计一个映射规则将属性 A 的值映射到 0 或者 1，依据此结果将元组划分到对应站点上实现关系间的站点依赖。

## 2.4 基于查询图的查询优化算法

### 2.4.1 基于链查询的 CHAIN 算法

CHAIN 算法是计算机图论里的一种算法，它是求图里最短路线的一种算法。在分布式数据库的应用当中，我们可以把站点和站点之间的连接关系用查询图来表示。找合并图形的最小代价，即关系连接的总代价时，可以把这个问题看成找这个图的最短路径，这其中的路径就相当于关系连接的代价。CHAIN 算法可以计算出基于链型结构的图的关系连接总代价最小，即它可以找到查询连接关系之间产生总代价最小的连接顺序，并且查询响应时间也最短的。

**算法思路：**CHAIN 算法的大概思路是先把查询语句转变成相应的查询图，根据查询图的图样结构从优化规则中，得出一个序列，这个序列就是这个查询的优化策略。

主要流程为：先从查询图的所有站点中找出其中 2 个站点使其 2 个站点合并之后所生成的连接代价最少，这里可以引用下 SDD\_1 算法的选择因子，根据选择因子找出这 2 个站点并且合并这 2 个站点；然后把合并之后的站点设置为根节点，再利用选择因子找出与根节点相连接的 2 个站点哪个连接代价最少，

使其与根节点合并；再把合并之后的节点设置为根节点，重复上面一个步骤，直到找不到任何站点为止；最后把上面步骤的序列记录下来，这个序列就是经过 CHAIN 算法把所要查询处理的优化处理序列。

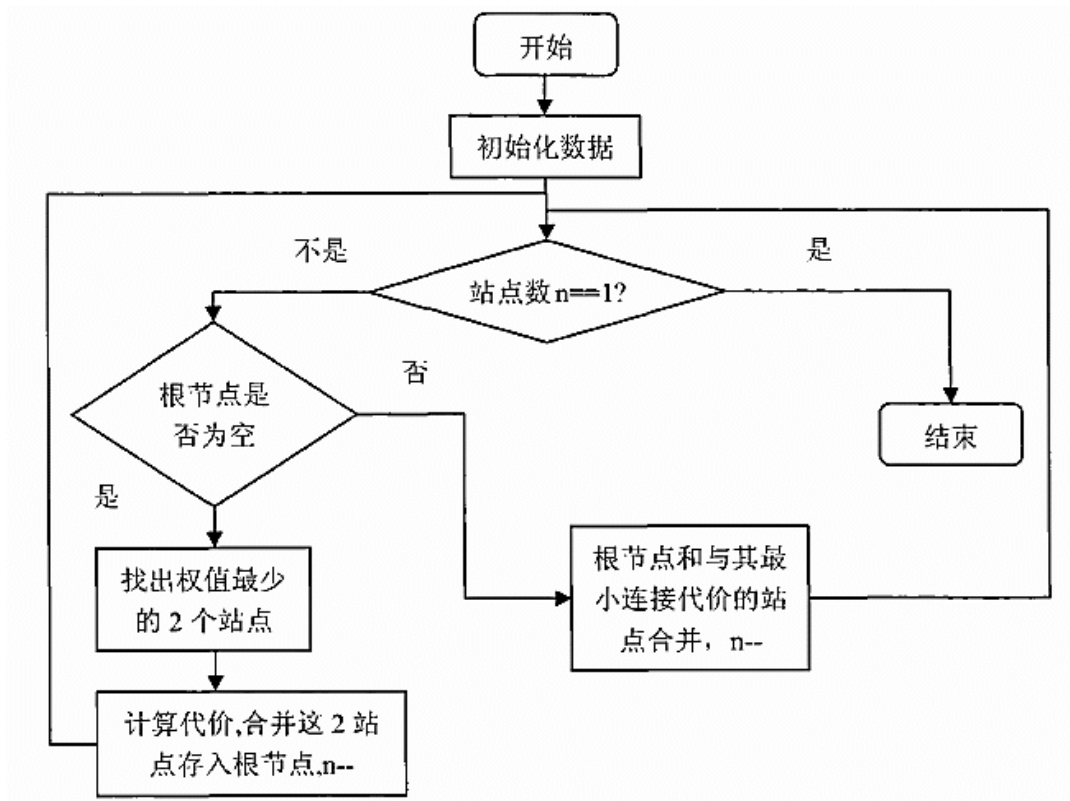


图 2-1 CHAIN 算法流程图

#### 2.4.2 Kruskal 启发式算法

如果查询图是非链型的，比如树型、星型等等，则它们的最小化查询响应时间问题是一个 NPC 问题。为了在保证计算量为多项式时间的前提下尽量接近精确值，学者提出了 Kruskal 启发式算法。

Kruskal 启发式算法是一个贪心算法，类似于图论中构造最小生成树的 Kruskal 算法的实现。它也是从一个带权图中找出最小代价生成树，这里的带权图可以看作是查询图，其中各边的权值就是两两关系间连接的代价。

**算法思路：**Kruskal 启发式算法的基本思想是：假设一个有  $n$  个顶点的查询图，首先计算查询图中所有边的权值。在计算的过程中，由于边上两端点表示做连接操作的两关系(包括中间结果关系)，根据各关系的元组数和元组大小，可计算出本地处理的代价 PC。

由于在边上的任何一个条件表达式都包含有关系间作等值连接的属性，根据各关系的  $\text{part}$  值，就可判断是否需要重哈希划分，从而计算出重哈希划分的代价  $\text{RC}$ 。按照哈希划分算法的代价公式，这时就可计算出两两关系间连接的总代价  $\text{Cost} = \text{PC} + \text{RC}$ ，即查询图中各边的权值。

选择其中一条具有最小权值的边，然后将该边的两端点  $R$  和  $S$  合并为一点  $U$ ，计算关系  $U$  的元组数  $|U|$  和元组大小  $|tu|$ ，设置  $\text{part}(U)$ ，同时记录连接  $R$  和  $S$  花费的代价  $\text{Cost}(\text{RS})$ 。当在选择一条最小权值边时，可能会出现两条以上的边具有相同的权值，为了使选出的边具有唯一性，算法在开始时就为查询语句中每个条件表达式  $\{R.A, S.B\}$  分配了一个参数值： $\text{number}\{R.A, S.B\}$ 。

Kruskal 启发式算法将选择候选边中  $\text{number}\{R.A, S.B\}$  值最大的那一条边。因为合并后的新结点中含有原关系的哈希划分属性，如果存在包含该属性的其它条件表达式，就可以利用它作为连接条件，从而避免了选择其它的条件表达式时在相应属性上做重哈希划分。

若关系  $R$  和  $S$  的元组数分别记为  $|R|$  和  $|S|$ ，元组大小分别记为  $|tR|$  和  $|tS|$ 。则有  $|U| = |R| \times |S|$ ， $|tu| = |tR| + |tS|$ 。如果  $A \in \text{part}(R)$ ，则  $\text{part}(R) = \{A\}$ ，否则  $\text{part}(R)$  不变；如果  $B \in \text{part}(S)$ ，则  $\text{part}(S) = \{B\}$ ，否则  $\text{part}(S)$  不变； $\text{part}(U) = \text{part}(R) \cup \text{part}(S)$ 。将  $U$  加入到查询图中，同时删去  $R$  和  $S$  及对应的边，再次选择一条最小权值的边，重复以上步骤，直到查询图中只剩下一个顶点为止。

---

### Kruskal 启发式算法

---

初始化：查询的总代价  $\text{Cost} \leftarrow 0$ ， $n \leftarrow$  查询图中顶点数

**while**  $n \geq 2$  **do**

    计算查询图中所有边的权值；

    选择其中一条具有最小权值的边；

    重构查询图：合并最小权值边的两端点  $R$  和  $S$  为点  $U$ ；

    计算关系  $U$  的元组数  $|U|$  和元组大小  $|tu|$ ，设置  $\text{part}(U)$  及  $\text{Cost}(\text{RS})$ ；

    计算查询连接的总代价  $\text{Cost} \leftarrow \text{Cost} + \text{Cost}(\text{RS})$ ；

**end while**

---

## 2.5 基于智能改进的查询优化算法

### 2.5.1 基于蚁群算法的查询优化

本质上蚁群算法属于进化算法中一种特殊的启发式全局优化算法，具有分布计算、启发式搜索和信息正反馈的特征，适合解决多维动态优化问题。在蚁群算法中，每一个个体都会通过释放信息素的方式对周边环境信息改变，而且个体也能够对周边环境的变化进行实时感知，通过环境实现相互通讯。

具体操作过程为：将连接操作进行拆分，寻找最优的查询方案即寻找最优的连接路径，即类比于蚁群问题中的寻找最优路径问题。在蚁群问题中通过代价估计模型计算连接代价作为蚁群算法中的路径权值，利用蚁群算法在寻找最优路径上的优点进行查询优化，在搜索过程中，利用分布式计算，可以实现多个个体并行计算，有助于提升计算能力和运行效率。

目前该解决方案仍处于研究阶段，但大量的实验结果已表明，该算法不仅有效，而且具有较好的寻优能力，当在连接元组数增多时依然有更好的表现，查询时间有效缩减。

目前衍生出来的比较常见的蚁群算法还包括基本蚁群算法 AS、最大最小蚁群算法 MMAS、最好最差蚁群算法 BWAS 等。

### 2.5.2 基于鱼群算法的查询优化

该算法和蚁群算法有相似之处，均借助于自然界中诸如蚂蚁、蜜蜂等群聚类动物群体内的动物自治体模式，构建相应的结构来重现动物感知环境变化时所做出的自我调节及快速适应的过程，将动物自治体的模式与结构、鱼类活动与分布式数据库特点相结合，从而提出了用于解决优化问题的人工鱼群算法。

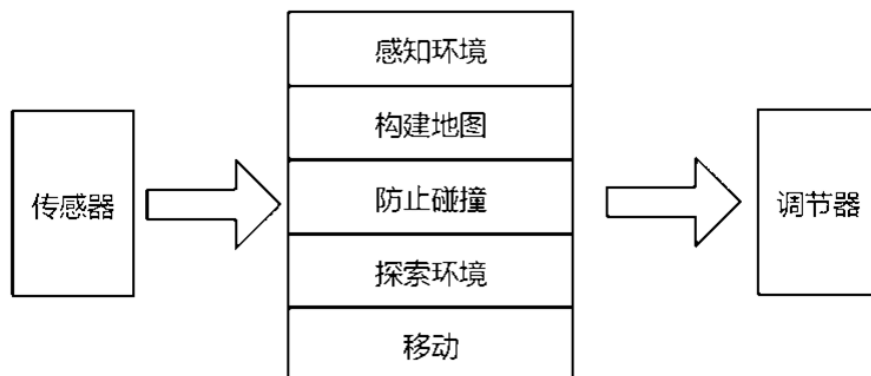


图 2-2 动物自治体结构



从简化分布式多连接查询的搜索空间入手，首先通过策略选择算法降低搜索空间的复杂度，然后采用该算法模拟鱼的觅食、聚群和追尾行为，利用它对极值敏感、反应迅速且同时具备较强的察觉并逃离局部极值的能力等优势在鱼群中构造单个人工鱼个体的局部寻优能力，最终达到全局最优状态即实现分布式数据库的最优多连接查询。

### 2.5.3 基于并行遗传-蚁群算法的查询优化

遗传算法（GA）是一种借鉴生物进化规律演化而成的随机化搜索方法，可以通过对自然进化过程的模拟，寻求最优解。其基本原理是生物学中的选择、交叉、变异和遗传等进化现象，本身具有自适应性和自学习型的特点，在大数据处理中有着良好的适用性，经常被用于解决最优化问题。

顾名思义，并行 GA-MMAS 是一个将遗传算法与蚁群算法结合并实现并行化查询处理的解决方案，它能够充分发挥分布式计算集群优势，缩短算法执行时间。算法思想为：先将种群分割成多个大小相等的子种群，保证每个线程独立完成子种群遗传操作，保留从其他种群获取的待交流个体，依照串行 GA 选择机制，选出进化父体，以本地进化父体与待交流个体完成子种群间交流。在预设进化次数完成后，主线程会对所有子种群的局部最优解进行归约，最终得到全局较优解，再经 MMAS 算法全局寻优，得到全局最优查询序列。

## 三、个人感悟与见解

### 3.1 学习感悟

在分布式数据库领域，查询优化始终是一个热门研究问题。本文对分布式数据库查询处理相关技术进行了研究，分析了分布式查询处理和优化的相关内容，涉及了分布式数据库的层次结构和分布式查询优化算法的分类。在讨论查询优化的算法中，综述了四类比较经典的分布式查询优化算法，在最后一类中对三种比较新的改进算法进行了简单的介绍。

关于这些方法的优缺点：首先，在广域网中由于数据的传输速率的限制导致通信代价的占比很高，此时通过缩减元组减少站点间数据传输量的半连接算法比较适用；在高速局域网或专线网络中以响应时间为优化目标，这种网络环境中通常以直接连接优化算法为主，包括使用 Hash 划分算法和分片复制等优化算法保障关系间的站点依赖。其次，基于查询图的优化算法根据查询操作转化为等价的查询图，按照相同划分属性合并规则不断合并迭代，保证最终查询图内的关系数量相当以及各子查询所需时间的均衡。最后，新型的智能算法未



来的发展前景尚需要时间来检验，相信人们在当今数据爆炸增长的时代对于速度的追求是无止境的！

### 3.2 个人见解

想要了解进而研究某一领域，需要做好三件事情，首先，要搞清楚这一领域到底是做什么的，然后，需要了解目前存在的尚未开始解决或者是尚未彻底解决的难题，最后，需要根据前人的研究提出一些自己认为能够解决这些难题的看法。

放到分布式数据库系统的查询优化领域来说，**第一个问题**关于它是做什么的，前面已经探讨过，所以不再赘述；**第二个问题**是分布式数据库系统查询优化有什么困难，为什么就解决不好。其实经过前面的探讨，发现困难还是很多的，相比于集中式数据库，分布式数据库数据的分散存储使得查询处理中需要它们在站点间相互传递，因此除了减少优化参数的搜索空间，还要考虑如何减少网络传输数据量，由此引申的还有数据应该如何储存（比如站点依赖和哈希可划分方式），还有如何选择搜索策略，如何更加准确地估计搜索代价，如何利用合理利用并行技术等等。

**第三个问题**是基于前人的经验提出自己独到的解决方案，由于这里仅仅是读了一些文献，参考了几篇简单的综述，因此提出解决方案是不可能了，但是可以简单地说一下见解。其实不难发现让分布式数据库系统在查询的时候产生的总代价减少，可以从三个方向来考虑，数据自身、查询计划以及计划的执行过程。

数据在集群上存储避免不了分区的问题，数据分区对数据库整体的性能影响十分巨大。并行性则是通过将数据划分为可以独立处理的子集来实现的，在分布式环境下实现并行化的操作对性能而言至关重要，但是数据的重新分区也是一个非常昂贵的操作，最小化数据分区操作的数量可以产生显著的性能改进，如何在数据分区操作与数据并行操作中达到一个平衡，对提升分布式环境下的查询性能有重要意义。

查询计划对查询性能的影响可以说是三者中最大的一个，关键因素仍然在于代价模型。随着分布式数据库的发展，通信模型、一致性算法、数据分片策略等因素等都会影响查询代价。

查询的执行过程对于性能的影响也十分显著。虽然查询的执行过程十分依赖于查询优化器选择的最终执行计划，但在执行过程中对底层的扫描方式或者连接方法，即逻辑操作符转换为物理操作符的过程中，往往存在许多的优化可

能，甚至数据的进一步筛选都能够从根本上为查询计划的选择提供更优的方案，从而减少整个查询执行的时间。

我觉得未来发展的方向一定是传统的查询优化技术和人工智能技术相结合，查询优化器和查询执行引擎协同工作，一些蚁群、遗传、模拟退火算法以及深度神经网络技术等都会为分布式数据库的查询优化打开新的大门。总之，在新的环境、新硬件和云服务及分布式的大势之下，查询优化领域一定会是一个不断探索与进步的领域，结合 NewSQL 这种新型的数据库相信也一定能够碰撞出更多的火花！

## 四、参考文献

- [1]龚浩. 分布式数据库查询处理和优化算法的研究[D].重庆大学,2005.
- [2]尤沛泉. 基于查询图的分布式数据库查询优化算法的研究与应用[D].长春理工大学,2011.
- [3]王宁. 分布式数据库查询优化的研究与实现[D].哈尔滨工业大学,2018.
- [4]茅潇潇. 分布式数据库并行连接查询的实现及优化[D].华东师范大学,2018.
- [5]樊敏. 基于分布式关系型数据库的查询算法优化[D].电子科技大学,2020.
- [6]聂小雄. 分布式数据库查询优化的研究与实现[D].电子科技大学,2020.