|  |
| --- |
|  |
| 分布式数据库终期报告 |
|  |
| **陈俊 2009210893 陈诚 2009210894 张腾 2009210958** |
|  |
| **2010-1-17** |

|  |
| --- |
|  |

# 1 网络通讯

网络模块在工程中主要由Global、Network两个package构成，其中Global主要包含了一些全局的消息类型定义，自定义的传输消息，以及托管接口的定义，Network主要包含了连接类的定义，消息处理类的定义，以及管理每个站点上连接的类的定义。网络部分主要实现的功能是四个站点间的数据通信，我们的工程中最终实现的网路结构为：



## 1.1站点连接

网络是整个分布式数据库中的一个底层的功能，在完成这次实验的过程中，我深刻的认识到网络模块设计对整个工程的重要性，所以为了保证网络模块的正确性，我们的网络模块是完全独立于分布式数据库中其他模块设计的。整个工程的连接过程为，初始化时四个站点并没有建立任何连接，而是每个站点都在特定端口起一个Socket监听，等待连接。然后由客户端登录到Control Site通过命令行进行设置，分别指定Site1，Site2，Site3对应的主机，设置完成后Control Site与三个站点建立连接，并且这个连接是持续的，当初始化工作结束后Control Site就通过这条连接向一个局部站点发送数据以及命令。而当一个局部站点需要向Control Site发送数据或者返回消息时，如果没有建立连接，则需要先建立连接再做发送操作，所以任意两个站点之间保持了两个Socket连接。

### 1.2网络模块的接口

网络模块的主要功能被封装到了一个叫做SocketConnector的Class中，提供了下面几个接口：

**sendString(String str, int siteType)**

**sendObject(Object obj, int siteType)**

分别用于发送一个字符串（例如用于显示的结果以及控制信息）和一个自定义的对象（必须是Serializable的）。

网络模块同时提供了一个用于获得SocketConnector的接口：

**getConnectorByAddr(String host, int port)**

在其他模块中调用可以方便的得到当前站点与输入站点的连接，然后可以利用上面列出的发送字符串或者发送对象的接口与对应站点完成通讯。

## 1.3消息处理

在每一个站点上，网络模块都会开启一个线程接收其他站点发送来的数据。我们的网络协议设计是发送站点在需要发送数据时直接发送，接收站点的接收线程一直处于接收状态，当接到数据后，并不向发送站点返回确认信息，发送的可靠性由TCP协议保证。为了提高通讯的效率，在接收线程中收到数据后并没有立即处理，而是把接收到的数据全部放到一个消息队列中，然后再启动一个事件处理的线程来分别处理接收到的数据，而具体的处理过程交给其他模块来托管，工程中定义了多个托管函数，当接收到不同数据时，会调用相应的托管函数，例如：

**didReceiveObject(int siteType,String host,int port,Object obj)**

**didReceiveString(int siteType,String host,int port,String str)**

托管函数在需要对接收到的数据进行处理的模块中实现。

## 1.4底层数据访问接口

为了方便在各个模块中访问底层数据库并简化数据处理的步骤，在工程中我们实现了一个访问底层数据的接口：

**executeSQL(String onSite, String toSite, String SQL)**

这个函数完成的功能是在onSite上执行SQL表示的SQL命令最后把执行的结果发送到站点toSite上。所以在整个程序初始化时，会自动与底层数据库做连接。实现了这个接口以后，对完成分布式数据库上层功能设计的同组其他成员提供了极大的便利。

## 2 Global Data Dictionary

全局数据字典（GDD）在整个分布式数据库中起着中心枢纽的作用。总的来说，有两个方面要用的全局数据字典。

首先是将一条全局的命令映射到相关的站点上。这包括要找出插入和删除命令涉及到的本地站点，以及一条全局的查询命令最后要分解到各个站点上去执行。

其次是要将一条全局的查询语句进行优化。优化的内容包括一元操作符的下移（比如选择和投影），全局查询树的剪枝（水平划分和垂直划分的剪枝）。

* 总体结构

为了方便对全局数字字典信息的查询和管理，整个的全局数据字典都被保存在一个总的类结构中，然后在这个总类中采用类包含的方法，把全局数字字典需要的信息都以类的形式包含进去。如下图所示，GDD中主要包含两大类的信息，一类是站点信息，每个站点都是一个GSite类的对象，其中存储了站点的IP地址，端口号，以及类型。如果是存储数据的站点还有其存储的表格信息。第二类是表格信息，里面包含了客户所创建的所有的表格信息，每个具体的表格都是一个GTable的对象，里面信息由表明，包含的属性名字和类型，分片类型以及分片的信息。具体每个分片信息又是一个GFragment类型的对象，由于GDD支持两种类型的划分：水平划分和垂直划分，因此具体的分片信息会根据划分的类型来存储。水平划分就会记录划分的条件，垂直划分就会记录划分的分片所拥有的属性。

图 GDD结构图

3 Parser

客户端使用的是命令行输入的方式，使用者在客户端输入命令之后，客户端就会把客户输入的命令直接传到控制站点，控制站点在收到用户端的命令之后就会调用parser对收到的命令进行解析。

Parser可接受的命令分为两大类，第一类命令是用来初始化数据库，以及显示信息的命令。第二类就是SQL语句，用来创建表格，插入，删除以及查询。当parser接受到客户端传来的命令之后，就对命令开始解析。然后根据命令的类型，分别调用不同的类来存储控制站点所需要的信息。比如说，当客户端输入表的划分信息的时候，parser就会从输入的命令之中解析出被划分的表的名字，表被划分的分段，划分的条件以及划分后的片段对应到的各个站点。客户端输入一条SQL select语句的时候，parser就会解析出select的对象，select的表格，以及select的条件。

当parser接受到一条无法识别的或者语法错误的命令的时候，就会简单的告诉客户端输入的命令有误，然后继续等着接受下一条语句。

# 4查询处理

查询处理模块主要是针对select语句进行的处理过程,大致分为以下几个过程：

1. 用户select语句到**全局查询树的生成**

这部分主要是对SELECT语句进行语句分析，将select中的查询元素转换成含有相关信息的二叉树节点的过程，主要是通过前一阶段的Parser处理得到。



查询二叉树节点的数据结构

1. **Query decomposition**

由于该系统默认的用户输入不会出现语法错误饿，因此系统没有实现语法错误检测和冗余消除。Query decomposition主要是在查询树中对一元操作符进行了下移处理。

1. **Query Localization**

根据GDD中各局部站点数据表的分布情况，将查询树中的逻辑关系表转换成局部站点关系表。

由于关系数据表经过水平或垂直划分，数据存放在不同的本地站点的，因此必须将查询树中的全局关系表用本地表替代。

规则：

|  |
| --- |
| IF 查询树中的关系表A是垂直划分（不妨设A=A1 JOIN A2）  If 只涉及到划分中的一个表A1或者A2，新建一个A1或A2节点替代原来的A节点：  Else  先新建A1和A2二个节点，然后新建JOIN节点，A1，A2作为JOIN节点的左右孩子节点，然后用JOIN节点替代原来的A节点，此时还要将涉及到原来A表的一元操作节点根据涉及到A表的属性进行下移,移到A1或A2所在分支下  Else  根据GDD中A表的分片信息建立多个局部节点，替代原来的A节点，新加入的局部站点作为上一局部站点的左孩子节点 |

1. **剪枝操作**

根据要求，系统必须能准确完成剪枝操作，其中主要是根据查询树中的一元操作（如id =1234，name=“USA“）去掉多余或者不满足条件的局部表节点。因为所有的非JOIN操作都下移到相关的表分支下，所以只需向上回溯遍历从表节点到JOIN节点之间的操作节点。其规则如下：

|  |
| --- |
| For 遍历与A表相关的操作节点B  If 操作节点B与A表中的局部站点A1中的分片信息有交集，保留A1，访问下一个局部站点A2  Else 删除局部站点A1，访问下一个局部站点A2 |

另外在涉及到join操作时也需要进行优化，因为join一边的表的操作节点可能是join另一边水平划分的判断条件，如对于Customer.id >12000 and customer.id=Orders.customerd\_id,查询树中的Customer.id >12000操作节点可以对Orders表进行优化，对于与customerd\_id>12000没有交集的局部站点可以删除。其规则如下：

|  |
| --- |
| for 遍历join二边的表的操作符节点  if该操作符节点中的属性为join操作中的属性如（id）且作为join涉及的另一表的划分条件，则删除无交集的局部表节点  else 访问下一个局部表节点直至访问完 |



查询优化的大致流程图

1. **优化处理**

优化处理包括二个方面：站点之间的数据传送量和join操作的优化。

由于时间仓促我们在优化方面做得不好。这二方面的优化主要是涉及join操作的。

站点数据传输规则如下：

|  |
| --- |
| If site1.A JOIN site2.B 并且A的数据量>B的数据量  将B传输给site1  Else  将A传给site2 |

对于join操作，为了提高join速度，主要采用并发操作，因为二个table A join B可能涉及多个本地表之间做JOIN操作，如可能会出现site1.A JOIN site2.B AND site3.A JOIN site3.B，这样可以再多个站点并发进行join操作，提高速度。另外对查询结果做join时，对join的属性建立索引，这样可以提高速度。

1. **查询优化树转换成本地SQL查询语句**

对于优化后的查询树，如何将查询树转换成查询语句涉及到对查询树的遍历问题。

由于前面已经将非join操作节点下移，因此上层节点均为JOIN节点，查询语句的生成过程如下：

|  |
| --- |
| If 查询树不涉及join操作  遍历查询树，生成查询语句，根据GDD局部表信息发送给相应站点  Else  （后序遍历查询树）  If 当前节点不是join操作节点  将该节点以下（包括该节点）的树分支转换成查询语句发送到相应表所在局部站点，并将查询结果返回给控制站点  Else  遍历左子树；  遍历右子树；  对join二边返回的查询结果进行join操作，返回操作结果 |

对生成的查询语句SQL通过调用executeSQL(String onSite, String toSite, String SQL)发送给toSite。

5总结

#### 5.1陈诚的总结

这次的大作业是我第一次接触如此大以及如此复杂的工程，尽管老师和助教一再提醒我们从早开始，但是从一开始，我们小组就没有认识到工程的复杂性，也没有任何的紧迫感，因此我们的进度一直很慢。而当我们意识到自己的工程以及非常落后的时候，期末的考试和各种作业接踵而来，使得原先计划的进度一拖再拖。弄到最后，不得不熬夜完成大作业。虽然最后的测试通过了，但是有很多东西都没有时间去实现。这对我来说，是一次深刻的教训。

通过这次的实验，我有3点体会。首先，对一个有一定规模的工程来说，一定要有一个合理的进度安排，以及能够按时完成进度。不然就只能熬夜交差了。其次，由于之前没有做过类似于分布式数据库的工程，因此并不了解小组成员之间的沟通的重要性。刚开始只是觉得，小组的沟通会很简单，对工程的影响也不会太大。但是，随着进度的往前推进，慢慢体会到小组沟通是一件非常重要也非常耗时的事情。因为讨论具体到某一个问题的时候，小组的三个人都有自己的想法，这时就需要在三个人之间进行权衡，就会有人放弃自己的想法而接纳别人的意见。这对于小组协作是非常重要的事情。第三点，通过这次大作业，分布式数据库的部分内容已经深入我的脑海，以后就很难忘掉了。

最后，很庆幸能够选择这门课程，感谢周老师，冯老师和范举助教的指导和帮助，让我获益良多。

##### 陈俊的总结

这可以说是我第一次做这么大的一个系统，之前没有项目经验。系统比较大涉及的方面也很多，尽管我们在课程初期就意识到这是一项艰巨的任务，也根据老师的意见进行了分工，我做的查询优化这一块。因为没有实际的项目经验，我花太多的时间思考查询的优化过程，而迟迟没有动手编写代码，事实上仅仅通过纸上谈兵很多问题都解决不了，通过和同学完成这个项目，我的第一感受就是刚开始不能把问题想得太复杂了，应该有简单到复杂，先实现简单的查询优化然后一步一步改进。由于空想没有带来根本的解决方法，我浪费了很多的时间。最后导致时间不够，优化做得不够好。另外一个问题是对join操作的优化问题，在实际系统能够真正运行之间我们其实对优化没有一个具体的感性认识，到最后真正运行程序时才发现join不优化的后果，当二个表数据很大时，join要花将近十分钟。原来考虑的站点之间的数据传输问题其实不重要了，最后的响应时间其实主要取决于join操作的优化。最后是作为一个团队需要有很好的协作，需要相互配合，这也是一次难得的经历。大家一起共同熬夜的经历深深印在我的脑海里。

#### 张腾的总结

这次实验中，给我留下印象最深刻的事情是，在实验检查的前一天晚上，我们组三个人一起为了将最后的个人完成的工作整合到一起，整个晚上都没有睡觉，虽然后来感觉很累，但是我们仍然都坚持到了最后，反复的调试，修改程序，终于最后可以正确的执行命令且显示完成结果了，当实验最终检查完成时，我们终于感觉到了前所未有的困意，几乎就要倒头大睡了。在完成整个实验设计的过程中，我们学到了很多关于分布式数据库的知识，但是让我最难忘的还是小组一起度过的那几段没有晚饭，没有睡眠，全力投入完成实验设计的时间，将在我研究生生活中打下深深的烙印。