分布式数据库管理系统

第五组实验报告

伍广宁　袁乐爽　马冬哲

2010-1-17

目　　录

1 最终用户需求分析

1.1 用例分析

图 1 显示了分布式数据库管理系统中所有的用例。本系统只有一种最终用户，既全局管理员。他既可以创建、删除、修改数据库，又可以查询数据库中的数据和状态等信息。

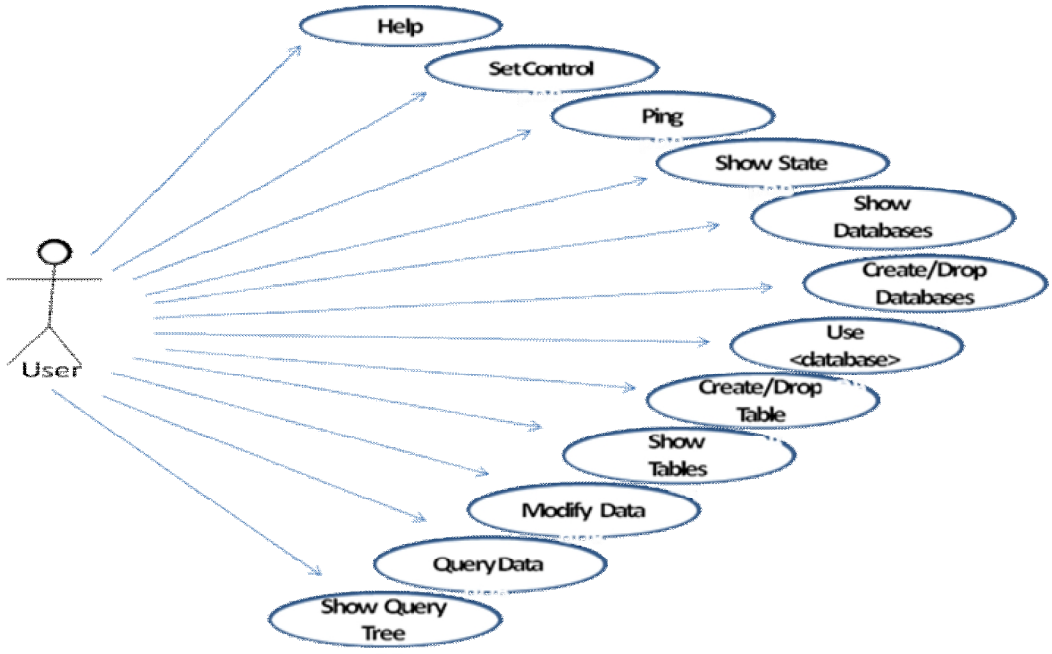

图 1 用例图

本系统共有 12 个主要功能：

- Help：显示所有可识别的命令列表；
- Set Control：指定执行 Control 功能的 Server；
- Ping：测试指定地址是否有效；
- Show State：显示 Client 端当前的状态；
- Show Databases：显示全局数据库列表；
- Create/Drop Database：创建/删除全局数据库；
- Use <database name>：指定当前所使用的全局数据库的名称；
- Create/Drop Table：在当前使用的数据库中创建/删除一个表；
- Show Tables：显示当前使用的数据库中的所有的表名；
- Modify Data：向表中插入或删除数据记录；

- Query Data：查询表中的数据记录；
- Show Query Tree：显示客户端上输入的最后一条 select 查询语句的查询树。

1.2 关键用例描述

1.2.1 用例 1：Create/Drop Database

- 用户必须能创建新的全局数据库或者删除一个已经存在的全局数据库。
- 如果用户创建了一个新的数据库，那么这个新数据库的信息要记录在 GDD 中；同样，如果一个用户删除了一个数据库，那 GDD 中的关于这个数据库的信息也要被删除。
- 用户新创建的数据库不应该与现有的数据库重名，否则创建失败，返回一个失败信息。

1.2.2 用例 2：Create/Drop Table

- 用户必须能在当前使用的数据库中创建一个新的表或者删除已经存在的一个表。
- 如果一个用户创建了一个新的表，这个表的信息要同时被保存到 GDD 中；同样，如果一个用户删除了一个表，那 GDD 中的关于这个表的信息同时也被删除。
- 用户创建的新表的名字不能与数据库中存在的表重名，否则建表不成功，会返回一个错误信息。
- 用户所建新表的表名要符合 MySQL 规范。
- 如果用户建表或删表成功，系统会返回一个操作成功消息。
- 用户在创建表的同时只要指定分片参数就可以对表进行分片，同时将分片信息记录在 GDD 中。

2 系统结构概要设计

2.1 三层概念模型结构

从概念上讲，本系统由三层结构组成（如图 2 所示）。最上层为客户端，中间层为控制服务器，最下层为局部服务器。

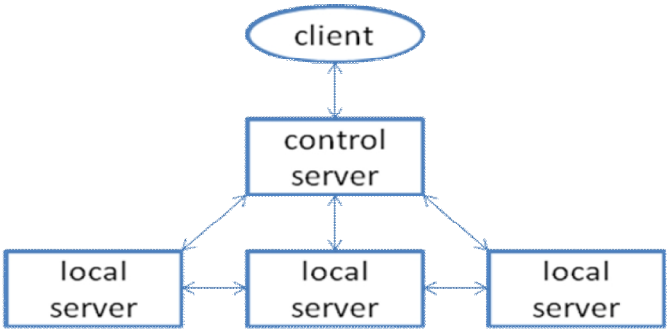

图 2 三层概念结构图

客户端通过 socket 协议访问控制服务器，向控制服务器发送数据库查询或者数据更新命令。

控制服务器在数据库管理系统中发挥非常重要的作用，它负责连接最终用户和局部数据库。最终用户只能和控制服务器进行交互。控制服务器接收到最终用户发送来的命令后，首先对命令进行解析，识别命令的类型，然后构造查询树并调用全局查询优化算法，最后生成一组执行序列并发送给相关的局部服务器。

虽然局部服务器不负责全局查询的处理及算法的优化，但是它负责执行和处理控制服务器或其他局部服务器发送过来的命令和数据，同时将执行结果发送给提前指定好的目的服务器。各个局部服务器之间可以相互通信。

图 2 中的三层结构是概念上的三层结构，在实际的分布式数据库管理系统中所有服务器端都可以完成控制服务器的任务，只不过在执行一条查询时只有一台服务器被指定执行控制服务器的功能。

2.2 三层结构组件

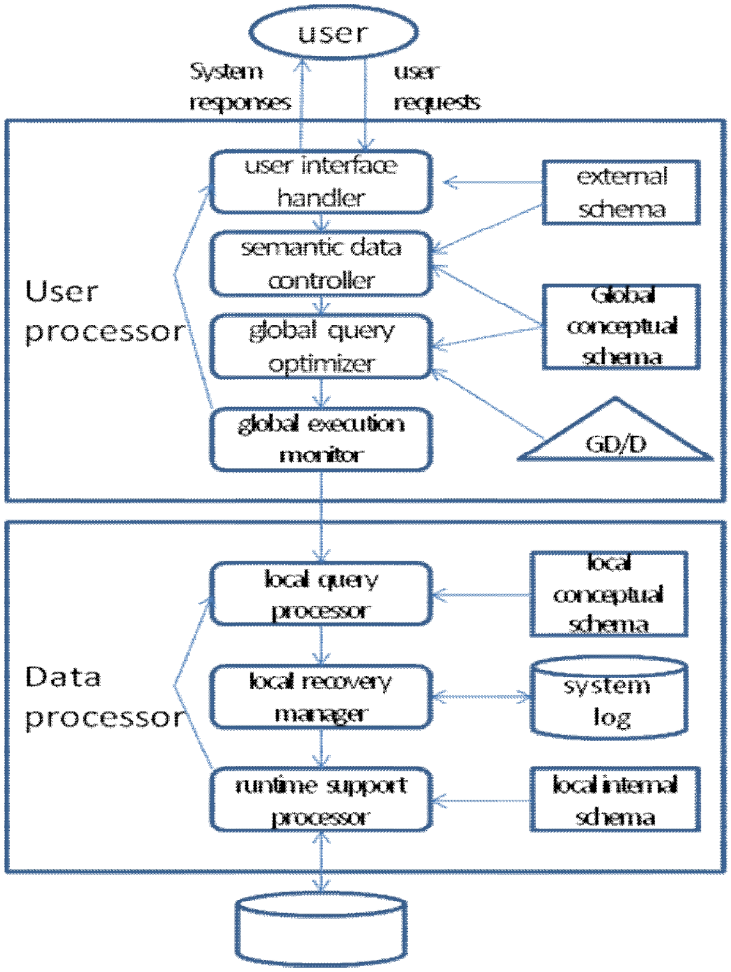

图 3 三层结构组件

图 3 显示了总体框架中各层结构的组件。从图中可以看出一共有两个组件，一个是 Control Server 中的组件 User Processor，另一个是 Local Server 的组件 Data Processor。

2.2.1 用户处理器

用户处理器是 Control Server 的组件，它由四个部分组成：

- User Interface Handler：当用户命令到达的时候，它主要负责解析用户命令；在将结果数据传给用户之前，它负责将数据规范化；
- Semantic Data Controller：它实现的功能主要是利用完整性约束和授权来检查用户的查询是否可以执行；
- Global Query Optimizer：决定一个查询的全局执行策略以形成最好

的查询算法，同时将全局查询转化为局部查询；

- Global Execution Monitor：通过与其他服务器的交互协调用户命令的分布式执行。

2.2.2 数据处理器

数据处理器是 Local Server 的组件，它由三个部分组成：

- Local Query Processor：对本地数据库上的数据进行查询更新等处理；
- Local Recovery Manager：保证局部数据库即使失败，数据仍然是一致的；
- Runtime Support Processor：与操作系统交互的接口。

3 系统结构详细设计

3.1 客户端

图 4 展示了客户端的详细结构设计，其中每个方框表示一个线程，虚线箭头表示线程间通信，实线箭头表示客户端与外界的通信。

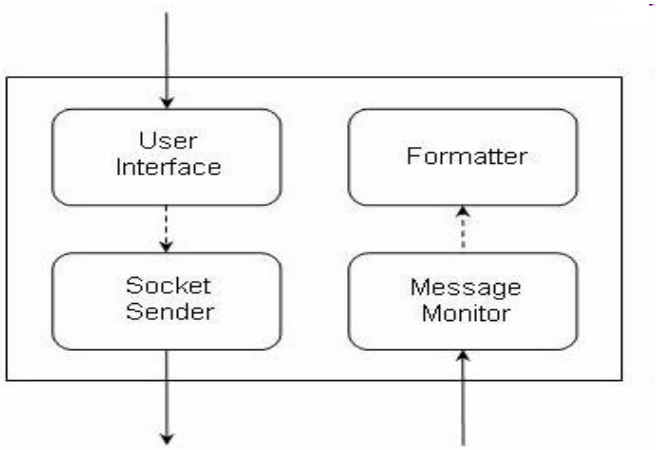

图 4 客户端结构详图

User Interface 首先通过读取配置文件或用户输入完成一些必要的配置，然后启动 Socket Sender 和 Message Monitor 两个模块，接着它等待用户输入，并将命令做一定的规范化，然后将其放到 Socket Sender 的任务队列里，再次等待用户输入命令。

Socket Sender 被创建后一直处于运行状态。当任务队列为空时，它会一直处于阻塞状态；当任务队列不空时，它会依次从任务队列里取出任务并将其发送给指定的目标服务器。

Message Monitor 被创建后也一直处以运行状态，它负责监听指定的端口。当收到一条消息时，它首先判断消息的类型。如果是没有格式的文本消息，则直接输出到屏幕上，否则创建一个 Formatter 线程，对消息进行格式化并输出。

Formatter 负责将有格式的数据（如表格）格式化，并输出到屏幕。

3.2 服务器端

图 5 展示了服务器端的详细结构设计，其中每个方框表示一个线程，虚线箭头表示线程间通信，实线箭头表示客户端与外界的通信，空心箭头表示各线程之

间的创建关系。

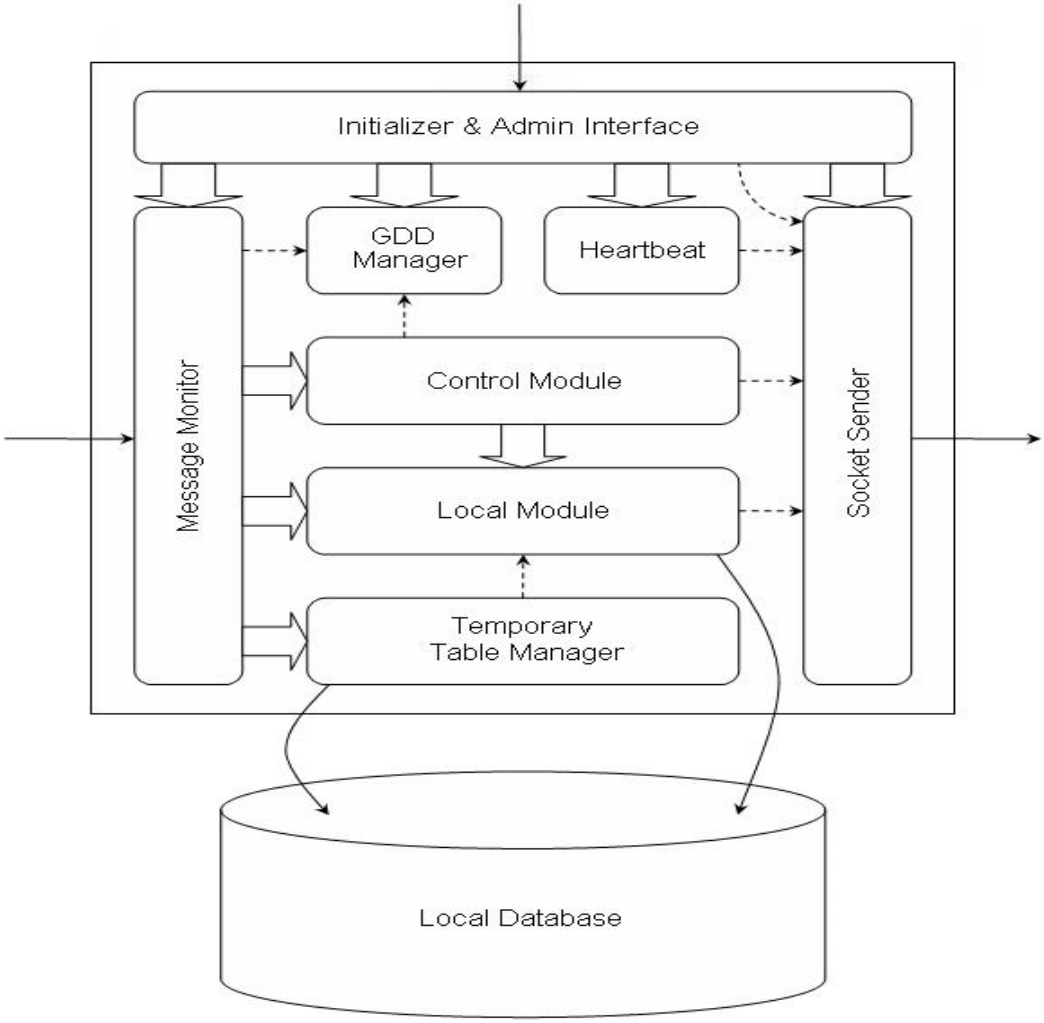

图 5 服务器端结构详图

服务器端各模块的设计遵循以下原则：

- 功能划分；
- 增强并行性；
- 避免同步 I/O。

服务器端 8 个模块中，只有 Control Module、Local Module 和 Temporary Table Manager 三个模块在需要时由相应的线程负责创建，可以同时存在多个运行实例，其它 5 个线程只在系统初始化时启动一次，并且一直处于可运行状态。

3.2.1 Initializer & Admin Interface

服务器启动以后，Initializer & Admin Interface 首先通过读取配置文件或用户输入完成一些必要的配置，然后启动 Message Monitor、GDD Manager、Heartbeat 和 Socket Sender 四个模块，接着试图同服务器列表里

的其它服务器同步，更新自己的 GDD 和服务器列表。最后，作为本地数据库管理员的输入接口，等待输入命令。

3.2.2 Socket Sender 和 Message Monitor

Socket Sender 和 Message Monitor 两个线程所实现的功能和客户端的 Socket Sender 和 Message Monitor 基本一致。Socket Sender 负责发送任务队列里的消息。Message Monitor 根据所接收到的消息的不同，启动相应的线程或将消息传递给相应线程。

3.2.3 GDD Manager

GDD Manager 实现三个功能：当有新服务器加入时，负责同步 GDD 和 Server 列表；收到其它 Server 的心跳时，更新 Server 列表中的时间戳；当全局 GDD 发生变化时，修改本地的 GDD。

当有新 Server 加入时，会在初始化的时候，由主线程分别向备选父 Server 列表里的每一个 Server 发送请求加入的消息，当某一已启动的 Server 的 GDD Manager 收到该消息后，向其发送 GDD 和 Server 列表。如果所有备选父 Server 均连接失败，则本 Server 作为 root 启动，生成空的 GDD 和只包含自己的 Server 列表。由于新 Server 的加入，Server 列表需要更新，这是由其它 Server 通过监听心跳来完成的。

Server 正常运行时，会通过 Heartbeat 线程以一定的周期向所有其它 Server 发出心跳，GDD Manager 收到心跳消息后，负责更新 Server 列表的时间截。若发现超时的 Server，需要向管理员做出提示。如果该 Server 未在列表中出现，则表明这是一个新加入的 Server，需要添加到本地的 Server 列表中。

Control Module 收到 Create/Drop Database/Table 消息后，会给相关 Server 的 Local Module 发送消息，请求修改全局数据库实例，同时给所有 Server 的 GDD Manager 发送消息，请求同步修改 GDD。新加入的 Server 需要从其父 Server 获得全局 GDD 拷贝，GDD 的传送也是通过一系列 GDD 修改命令来实现的。各个站点都会在永久存储器中存储一份 GDD 的镜像，其中的 GDD 修改命令可以周期性的通过扫描 GDD 来重新生成，避免过多无用的操作。

3.2.4 Heartbeat

Heartbeat 每隔一定周期向其它站点发送一次心跳。同时，Heartbeat 每

隔一定心跳次数会判断一次本地分片是否发生变化。如果有变化，它会重新计算全局优化时需要用到的统计信息，如记录条数、每个字段的最值、不同值的个数等，并发送给其它站点。

3.2.5 Control Module

Control Module 线程只有在被指定为控制服务器的机器上才能被创建用来做全局算法决定，它所接收的命令全部来自客户端。

- create/drop database：扫描 Server 列表，向所有 Server 的 GDD Manager 发送消息请求同步 GDD，向所有 Server 的 Local Module 发送消息请求创建/删除本地的全局数据库实例。
- show databases：扫描 GDD，将全局数据库列表以表格形式发给 Client。
- create/drop tables：扫描 Server 列表，向所有 Server 的 GDD Manager 发送消息请求同步 GDD，向相关 Server 的 Local Module 发送消息请求创建/删除本地的关系实例。
- show tables：扫描 GDD，将当前数据库中的关系列表以表格形式发给 Client。
- insert into：读取 GDD 的相关项，将数据分解，分别发送给相关 Server 的 Local Module，请求插入数据。
- delete from：读取 GDD 的相关项，将相关 Server 利用 where 子句进行剪枝，再向相关 Server 的 Local Module 发送消息请求删除相应数据。
- select：读取缓冲，若命中则直接使用执行序列，否则读取 GDD 的相关项，构造查询树，并进行优化剪枝，再根据优化后的查询树生成执行序列，然后判断相关的 Server 是否都在线（如果不是，用 INFO：消息通知 Client 查询无法执行），最后将执行序列分发给相关 Server 的 Local Module。注意在 Server 上需要缓冲规范化后的 select 语句以及优化后的查询树，以备再次进行查询或要求显示查询树时使用。另外 GDD 发生变化时应从缓冲中删除受到影响的查询树和执行序列。
- show querytree：读取缓冲，若命中则直接将相应的查询树发送给 Client，否则重新构造查询树，并进行优化剪枝。
- show plan：读取缓冲，若命中则直接将相应的执行计划发送给 Client，否则重新构造查询树，进行优化剪枝，并生成执行计划。
- show transfer：收集各个站点上某一查询的消息传递统计信息，并发送给 Client。

3.2.6 Local Module

Local Module 线程的功能是按照执行计划，对本地数据库进行创建、删除、查询等操作。Local Module 可能是由 Message Monitor 在收到一条消息后创建的，也可能是由本地的 Control Module 创建的。对于来自本地的命令不计网络传输量。任何一个 Local Module 线程被创建都会创建网络传输量的统计变量（可能与 Temporary Table Manager 竞争），而在它结束之前，会负责删除自己创建的统计变量。同时 Local Module 在结束之前，会删除自己用到的临时表。

3.2.7 Temporary Table Manager

Temporary Table Manager 线程接收来自其它站点的 Local Module 的中间表，将其插入到本地数据库中，并通知本地 Local Module 相关数据已准备好。它不发送任何消息。Temporary Table Manager 线程启动后，也要创建网络传输量的统计变量（可能与 Local Module 线程竞争），这些变量由相应的 Local Module 线程负责删除。

4 全局查询处理

4.1 查询树

全局查询的处理基于一个重要的数据结构：查询树。它将查询形式化为一个树形结构并在其上进行操作。本系统实现的查询树有以下特征：

- 共有三种类型的节点。分别表示基本表（Table Node）、连接操作（Join Node）和并操作（Union Node）。投影和选择操作在每个结点中作为属性存在（Union Node 上这两个属性始终为空）。这样做的好处是每个结点都可以被看成一个表（本地表或临时表）。在进行全局优化和制定执行计划时，可以为每个非叶子结点指定一个表名和生成站点；
- Join Node 含有两个子结点，分别表示 join 操作的两个子表；
- Union Node 可含有多个子结点；
- Table Node 一定是叶结点。

4.2 查询处理过程

当 Control Module 收到一条全局查询时，首先对查询进行标准化，如查询所有不带表名前缀的属性所在的关系、将选择操作的属性域前置等。然后，按以下步骤构造查询树、执行查询优化算法并制定执行计划。

- 构造全局查询树。叶子结点为 from 子句涉及到的全局表，非叶子结点为 where 子句中的连接运算，select 子句中的属性作为根结点的投影操作，where 子句中的选择操作作为叶子结点上的选择操作。
- 定位分片信息。扫描 GDD，利用相应全局表的 makeup 信息，将所有叶子结点替换成它的 makeup 子树。由于本系统将水平划分、垂直划分和混合划分进行了统一的表示，所以该步骤不需要考虑分片的类型。
- 投影下移。从根结点开始遍历查询树，将根结点的投影属性和上层结点用到的连接属性传播到下层结点，作为投影属性。
- Union 上移。遍历查询树，若存在 Union Node 的父结点（可能不是直接父结点）是 Join Node，则将 Union Node 的每个子结点与 Join Node 的另一个子结点分别进行 join 运算，最后再 union 在一起。注意虽然经过该步骤后，原查询树中的一个结点可能参与到两次运算中，但本系统实现的查询树仍然只含有一个结点，这可以避免同一结点的重复计算，也可以避免不必要的网络传输。实际上，经过此步骤，查询树可能已经

变成了一个 DAG。

- 剪枝。根据剪枝算法，将计算结果必为空的子树裁剪掉。
- 替换查询树中所有属性的临时表名，计算每个叶子结点的估计大小，为制定传输策略做准备。
- 制定传输策略。后序遍历查询树，为每一个结点制定执行站点。其中，本地表在本地完成。对 Union Node 不进行处理，直接传输到需要进行 join 的站点，除非它是根结点。对 Join Node 根据子结点的不同情况查找一个传输量最低的站点。
- 制定执行计划。再次遍历查询树，根据各个结点的计算站点，生成不同站点的执行序列。

4.3 剪枝算法

剪枝算法基于 3 个基本原则：

- 如果某个分片的生成谓词与查询条件相矛盾，该分片被去除；
- 如果某个分片的全部属性与查询的投影属性无关，该分片被去除；
- 如果某个 Join Node 的两个输入含有矛盾的谓词，整个 join 分支被去除。

执行时，后序遍历查询树，利用 GDD 中的分片信息逐个结点判断是否需要剪枝。

4.4 优化算法

本系统将局部优化任务交给 MySQL 完成。全局优化由 Control Module 完成。

全局优化算法的基本思想有两个：一是提高查询执行的并行度，二是降低网络传输量。

- 为了提高查询执行的并行度，将 Union Node 上移至根节点，这样不但可以避免无用的计算，也可以将 join 操作分散到不同的站点并发执行。最后，将不同分支的计算结果汇总到一个站点上进行 union 操作。
- 为了降低网络传输量，本系统利用 GDD 中的统计信息来估算各种传输策略的代价，并选择最优的传输策略。
- 在生成执行计划后，对所有的 wait 操作进行了后移，即任何一个 Local Module 只有在需要使用一个临时表时，才会去判断、等待这个临时表。

5 任务分工

本次课程实验各阶段的任务分工如下表所示（按任务量由多到少排序）：

表 1 任务分工

阶段	分工
系统设计	马冬哲、伍广宁、袁乐爽、其他
消息及命令格式的制定	马冬哲
辅助模块的开发与测试	马冬哲、伍广宁、袁乐爽
基本操作的执行	伍广宁、马冬哲
全局查询与优化	伍广宁、马冬哲
基准测试数据的生成	袁乐爽
集成测试	马冬哲、伍广宁、袁乐爽
文档撰写	袁乐爽、伍广宁、马冬哲

6 总结

6.1 伍广宁的总结

这是我第一次全程参与到这样一个复杂的项目。在选课之前。已有很多朋友提醒我这门课并不容易。但当时我觉得只要足够投入，这些将不会是问题，而且我也很期望通过这样一个项目来让自己有所成长。

随后事情的发展渐渐偏离了轨道。马冬哲在系统框架上做了很多的工作，这从很大程度消除了我对于大作业的担心，但同时也让我变得大意。另一方面，我们的分工存在一定的混淆，比如我一直以为查询优化会由大家一起实现，所以迟迟没有开工。直到 12 月底，我们仍然没有一个基本的原型系统。于是最后不得不日夜赶工，2010 年的前几天都献给了大作业，即便如此，我们也未能如期完成，最后评测出现了性能上的巨大问题。

尽管如此，我从中收获了很多东西，我希望能做一个简单的罗列：

1) 交流非常非常非常重要；
2) 分工和计划一定要明确；
3) 将主要的精力投入在最重要的事情上；
4) 尽快实现原型，然后再去完善它；
5) 对于新手，调试远比编码要耗时间；

……

这个过程也让我认识到搭建一个合格的分布式数据库系统是一项多么庞大的工作。事实上，我们的大作业只关注了其中非常小的一部分内容，很多问题都被简单的忽略掉了。总而言之，这是一段令我难忘的回忆。我犯了很多错误，唯一可以安慰自己的是以后我有更大的把握避免这些错误。或许这就是成长。

6.2 袁乐爽的总结

在选课之前听很多师兄师姐说起这门课，说是这门能学到很多东西。我当时还没有太在意，觉得不就是一个课设嘛，没有太上心，后来加之马冬哲做了很多初级设计工作，感觉应该没什么问题（因为在本科的时候做课设最费事的就是做前期的设计），就滋生了一些大意。可是到了后来真正做起来的时候还真是问题频出啊，多亏这之中不管是从文档上还是代码上马冬哲同学都给了我很多帮助，非常感谢。从这次的课设中我学到了很多东西：团队的合作非常重要，不经历这

么复杂的项目真的不知道交流在实施过程中的重要性，其实我们这次的工作分工及其接口定义并不是很理想，以至于后来问题频出，这也是我们受教最深的地方，也许只有经历了这些问题，才能使我们对它的认识更深刻，可以使我们以后更少的犯同类的错误；通过这次的工作，我对分布式数据库有了一个更深刻的了解。

6.3 马冬哲的总结

本次课程实验的最终测评结果不是很理想，我要负主要责任。

首先，在初期设计阶段和基本的开发工作完成后，没有积极地与大家保持交流、督促工程的进展，造成时间表严重的推迟，以致后期把主要精力都放在调试上，没有太多的时间进行测试和优化。

其次，没有做好调研，掌握 MySQL 的基本特性，误导组员使用了负载很高的操作，造成最终测评结果不是很理想。而且在测试阶段本有机会发现这些问题，由于一些原因，没有能够把握这些机会。这些，都是由我的知识储备不足引起的。

第三，在测评之前，没有调试好测试环境，系统中原有的一些垃圾数据造成程序的异常，配置文件与执行脚本也没有准备好，这些失误造成不少麻烦，影响了测评人员的主观印象。

总之，与其它组相比，我们的系统做得不是很好，我感到十分抱歉，尽管大家都很努力。其中，伍广宁提出的查询树上 Union Node 上的选择和投影属性始终为空这一“定理”解决了我心中“union 上移”操作如何进行的难题，他提出的把查询树变成 DAG 的思路也很新颖，使我深受启发。袁乐爽也很用功，为了本次实验，她查阅、学习了很多资料，还不厌其烦地修改、优化她的代码。

通过本次课程实验，我也学到了很多东西—在设计阶段需要考虑哪些问题才能把系统设计得高效、可靠，在编写分布式、多线程的程序时可能遇到哪些问题，任务调研的重要性，严格遵守时间表的重要性……

最后，感谢周老师和冯老师一个学期辛苦的外语授课，感谢范举助教的悉心指导，感谢为我答疑解惑的其他师兄，感谢同样很努力的伍广宁和袁乐爽两位同学……