CIMS，Tsinghua University

分布式数据库系统

实验总结报告

黄科满(2009310522)
来洪波(2009211116)
辛　乐(2009310523)

2010-1-1

目录

分布式数据库系统设计及报告

1.功能描述与用例分析

系统支持的命令以及需要实现的功能：

1）初始化 GDD 命令

定义站点：定义 4 个站点，说明每个站点的 IP 和端口

site 4

166.111.180.71 3000

166.111.180.71 3001

166.111.180.71 3002

166.111.180.71 3003

定义数据库：数据库名称唯一

Market

定义数据表：同一数据库中的表名唯一，列的类型支持 int，char 和 varchar 三种类型

table 4

Customer(id int key, name char(25), rank int)

Publisher(id int key, name char(100), nation char(3))

Book(id int key, title char(100), authors char(200), publisher_id int, copies int)

Orders(customer_id int, book_id int, quantity int)

定义分片：将四个表分别根据要求进行分片

fragment 4

Book hf 3

id < 205000

id >= 205000 and id < 210000

id >= 210000

Publisher hf 4

id < 104000 and nation = 'PRC'

id < 104000 and nation = 'USA'

id >= 104000 and nation = 'PRC'

id >= 104000 and nation = 'USA'

Customer vf 2

id name

id rank

Orders hf 4

customer_id < 307000 and book_id < 215000

customer_id < 307000 and book_id >= 215000

customer_id >= 307000 and book_id < 215000

customer_id >= 307000 and book_id >= 215000

2）初始化站点数据库

站点分配：定义在每个站点中存储的数据表的分片

allocation 4

Publisher_1 Book_1 Customer_1 Orders_1

Publisher_2 Book_2 Customer_2 Orders_2

Publisher_3 Book_3 Orders_3

Publisher_4 Orders_4

3）导入数据

每个 txt 为一个数据表的所有记录，可以直接通过对话框进行选择。要求文件名为对应的数据表的表名。

4）插入数据

要求命令语句符合 SQL 语句的语法规范。对命令进行分析并插入到对应的数据表中。

insert into Customer (id, name, gender, rank) values (100010, 'Xiaoming', 'M',1)

5）删除命令

要求命令语句符合 SQL 语句的语法规范。对命令进行分析并删除对应的数据表中的数据。

delete from Customer where Customer.name = 'Marry'

6）查询记录

显示查询树，返回查询的结果，以及总的记录条数，并将查询结果显示出来

select customer_id,quantity from Orders where quantity < 8

2.系统用例分析以及执行逻辑分析

1) 在三台机器上启动三个 server 端，打开监听功能，用来接受命令，并将命令解析成为 SQL 数据库命令并执行。在一台机器上启动客户端，作为命令输入以及结果显示的界面。

（初始化，主要工作在于建立相应的 GDD）

2) 导入全局信息（GDD）。包括分布式数据库划分的方式，fragmen 存储的位置，各个分片的统计数据。

（客户端操作）

3) 输入命令行。运行客户端，启动全局计时器进行计时。

4) 客户端解析命令行，将命令行解析形成操作树节点。

树节点主要包括以下的几种类型：1）叶子节点，表示命令设计的表，为一 class。2）操作节点，包括选择（select），映射（project），连接（join）三种类型。每种操作节点设计为一 class。每种操作对应一种 SQL 语句。

5) 验错。检查输入的命令行是否出错。

6) 生成操作树。根据命令行，将节点连接成为操作树。这里操作树仅仅存在于内存，是一种形式化的表示。

（操作树的重写以及优化）

7) 重写操作树。根据优化算法对操作数进行优化，包括将 join 操作以及 select 操作进行合理的操作顺序的更改，根据全局信息进行操作的重写，形成新的操作树。

8) 全局优化。根据全局信息以及重写的优化操作树，将各个节点解析成为 SQL 命令，确定命令执行的顺序。包括整个操作树对应的操作动作，每个操作动作的先后顺序，执行操作的站点以及数据在不同站点之间传输的顺序和方式。

（以下为网络传输部分，需要设计相应的协议如命令协议以及数据协议，包括协议的命令结构，传输的方式以及传输的类型，传输数据的大小等）

9) 将命令封装成为 HTTP 协议包。将协议包根据命令与对应的 server 端建立连接，并将协议包发送给 server 端。

10) 接收解析命令。Server 端接收相应的消息，根据消息解析形成对应的 SQL 命令行，对当地的 MySQL 数据库进行操作。

11) 本地数据库操作以及优化。如果操作只涉及本地站点，获取相应的数据，将数据进行封装并按照命令和相应的站点建立连接传输数据。如果需要获得其他站点的数据，完成本地操作以后，向相应的站点发送消息建立连接获取数据进行操作。在此过程中需要在数据库上生成相应的中间表。完成本地操作以后将相应的结果传输给控制端或者是对应的站点，并删除中间表。

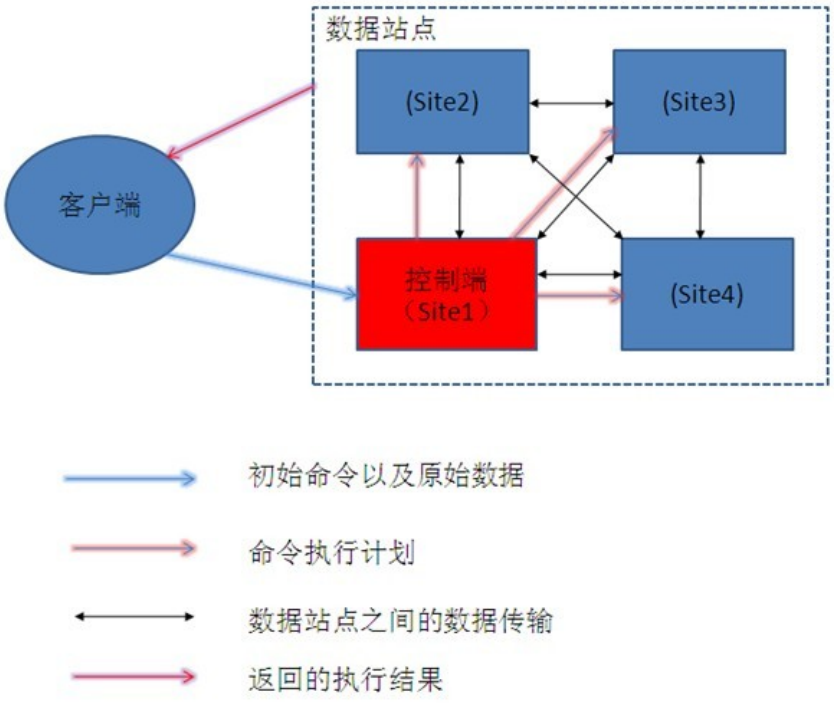
（以下为客户端显示部分）

12) 客户端完成数据接收。启动显示计时器。显示获取的数据列表。

13) 数据展示结束，停止两个计时器，输出显示所需时间以及整个操作所需的时间。将形成的优化操作数按照规则输出。

3.总体设计

为了实现以上的功能，系统功能主要分成三个部分：客户端、控制端和数据端。站点之间通过 HTTP 通讯协议进行连接。客户端负责为用户提供界面，接受用户指令并显示运行结果，将初始数据以及信息传送给控制端。控制端负责生成 GDD，解析命令并生成执行计划，向各个站点发送执行计划。数据站点根据执行计划执行命令，并将最终的结果返回给客户端显示。整体系统框架如下所示：

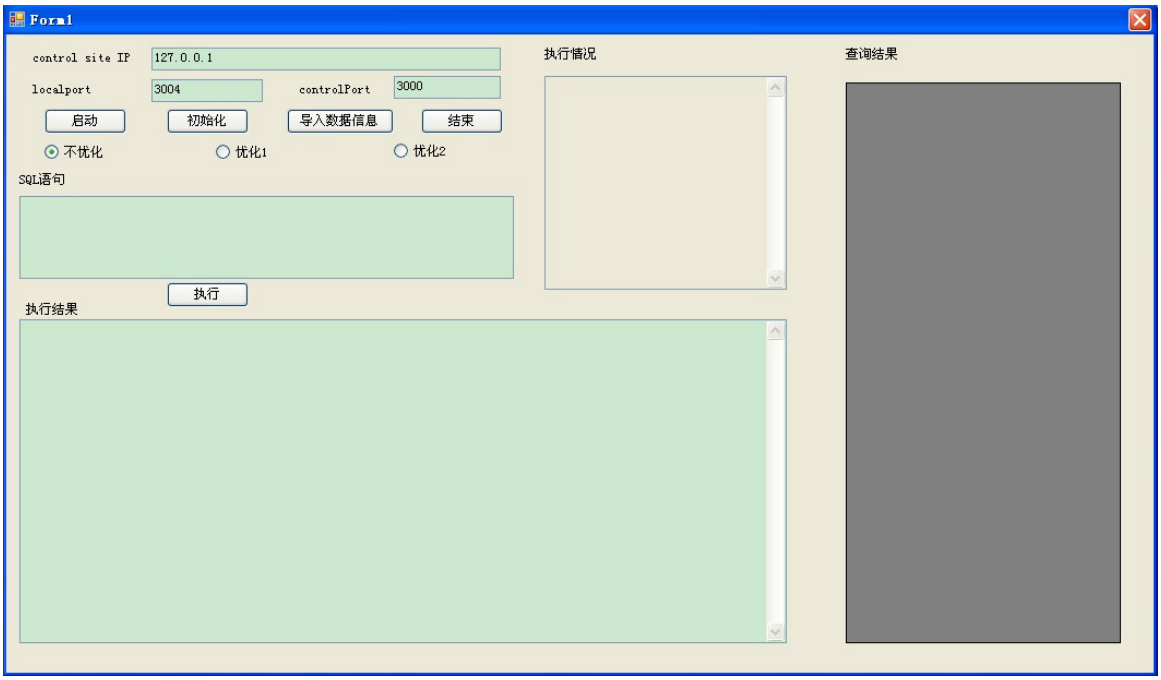


图表 1 数据库系统整体设计

4.模块设计

4.1 客户端

我们为客户端设计了较易操作的图形界面，用以支持数据库操作中的各项操作。客户端界面如下图所示。



完成的功能包括：

（1）设置控制端 IP 及端口、本地端口。通过文本框输入，进行读入和初始化设置；

（2）启动监听。在已获取控制端地址之后单击“启动”开始对四个 server 的监听；

（3）初始化。单击“初始化”按钮，在弹出的文件选择对话框中读入包含初始化信息的文本文件，通过对其中内容的分析，进行系统的初始化；

（4）导入数据信息。数据信息写在四个文本文件之中，分别是数据字段的值。单击“导入数据信息”按钮，在弹出的文件选择对话框中选择相应文件，进行数据导入。

（5）设置优化类型，分别对应不优化、优化类型 1、优化类型 2；

（6）读入 SQL 语句。在文本框中输入之后单击“执行”。

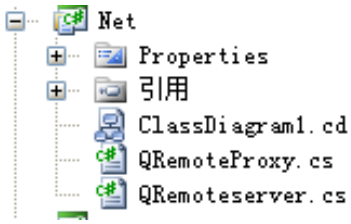
（7）结果显示。分三个部分：“执行情况”显示诸如“初始化成功!”“数据导入成功!”的提示语句；“执行结果”显示生成的查询优化树；“查询结果”展示查询之后的数据表。

4.2 通讯

通讯模块是分布式系统的重要组成部分。在该系统中采用 HTTP 协议作为通讯协议，采用请求/回复的方式实现站点之间的通讯。主要包含以下的两个部分：

1） 站点之间的数据传输和接受

主要包含两个部分：数据发送以及数据接受。通过 QRemoteProxy 和 QRemoteServer 两个类来实现。

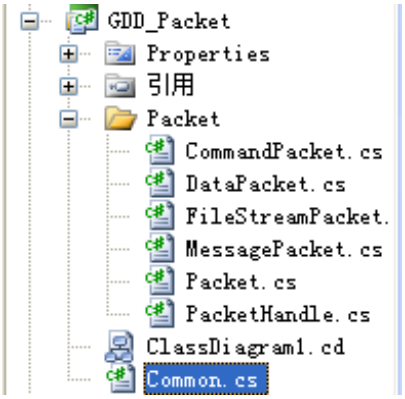


QRemoteProxy 类将需要传输的包进行序列化和压缩，建立于指定站点的连接，并将压缩完的数据发送出去。

QRemoteserver 类从连接的 Socket 中获得数据流，并解压缩，反序列化成为命令包。

2） 传输数据的相关定义

为了能够实现对数据库系统的处理，定义以下的命令包以及命令包类型：



Packet：虚拟类，是以下各种命令包的基类。

CommandPacket：命令包，存储执行的命令计划。

DataPacket：数据包，存储返回的数据包

FileStreamPacket：文件包，存储在各个站点之间传输的数据，采用文件存储的方式进行

MessagePacket：消息包，存储站点执行命令的结果，对站点执行情况进行监控

PacketHandle：提供对各种 Packet 进行序列化，反序列化，压缩，解压缩的方法

Common：公共类，定义了全局变量，包括命令包的类型，文件包类型，命令类型等公共信息。

```
//定义最大的缓冲区大小
public static int MAXBUFFET = 4096;
//定义数据包的类型
```

```
public static string filePacket = "FileStreamPacket";
public static string dataPacket = "DataPacket";
public static string messagePacket = "MessagePacket";
public static string commandPacket = "CommandPacket";
//定义命令包的类型
public static string InitializationCommand = "InitializationCommand";
public static string InsertCommand = "InsertCommand";
public static string DeleteCommand = "DeleteCommand";
public static string SelectCommand = "SelectCommand";
public static string OperationCommand = "OperationCommand";
//定义命令关键字
public static string createSQL = "create";
public static string defineSite = "define";
public static string sendData = "SEND";
public static string waitData = "WAIT";
public static string unionTable = "Union";
public static string joinOperation = "Join";
public static string selectOperation = "Select";
public static string projectOperation = "Project";
public static string insertSQL = "Insert";
public static string deleteSQL = "Delete";
public static string sendDataToClient = "SendData";
//定义文件包类型
public static string ImportFilePacket = "ImportFilePacket";
public static string InsertFilePacket = "InsertFilePacket";
public static string InitiaFilePacket = "InitializationFilePacket";
```

4.3 控制站点

控制站点是整个系统的灵魂，实现接收数据，命令解析，数据的分包，创建查询树，进行查询优化，生成命令执行计划，发送数据和命令包等功能，是系统的中枢。控制站点接受的包类型有以下的两种：

1） FileStreamPacket：文件包

ImportFilePacket 为导入的初始数据，需要控制端根据 GDD 的信息进行分包

InitializationFilePacket 为初始化信息，主要用来生成 GDD，并根据 GDD 的内容初始化各个站点并且在各个站点上建表

InsertFilePacket 则为直接导入的数据

2） CommandPacket：命令包

"InsertCommand";插入命令，对插入命令进行解析，分析需要执行的站点，并生成执行命令，发送到指定站点执行。

"DeleteCommand";删除命令，对删除命令进行解析，分析需要执行的站点，并生成执行命令，发送到指定站点执行。

"SelectCommand";查询命令，对接收到的查询命令进行解析，生成查询树，对查询树进行优化，生成查询命令执行计划，发送到指定站点执行。

"OperationCommand";本地执行的命令，根据命令类型进行执行。执行逻辑见数据站点。

4.4 数据站点

数据站点根据接收到的命令执行计划，连接本地数据库进行执行。在本系统中，数据站点接收到的执行类型主要为以下几类：

1）"create";在本地数据库中建立数据表，命令格式如下所示：

create|createSQL

2）"define";在本地数据库中定义数据库，命令格式如下所示：

define|defineSQL

3）"SEND";将数据表发送到指定站点。需要将数据表的结构以及数据都发送到指定站点，并删除中间数据集。命令格式如下所示：

SEND|TableName|SiteName|SitePort

4）"WAIT";等待从某个站点传送过来的数据，命令格式如下所示：

WAIT|TableName|SiteName|SitePort

5）"Union";将数据库中几个数据表合并成为一个数据表，命令格式如下所示：

Union|TableName1*TableName2*TableName3.....|ObjectionTable|TableField

6）"Join";对两个数据表进行Join操作，命令格式如下所示：

Join|TableName1*TableName2|TableName1.Key == TableName2.Key|TableName3|TableName3Field

7）"Select";从数据库中获得查询结果，存储成为中间数据表，命令格式如下：

Select|selectSQL|TableName|TableNameField

8）"Project";对数据表进行Project操作，命令格式如下所示：

Project|selectSQL|TableName|TableNameField

9）"Insert";将数据插入到指定数据库中，命令格式如下所示：

Insert|InsertSQl|TableName

10）"Delete";将数据记录从指定数据表中删除，命令格式如下所示：

Delete|DeleteSQl|TableName

11）"SendData";将执行结果以DataPacket的方式发送到客户端，命令格式如下所示：

SendData|TableName|SiteName|SitePort

为了支持以上的命令执行过程，为系统提供了一个LocalOperation类，封装了与本地数据库的直接操作，有效的实现了执行逻辑与具体数据库操作的分离。

4.5 GDD 设计

GDD 的设计主要包括两个方面，一个是数据库的有关信息，包括数据库、表、字段、站点等的设计。而是表的分片信息的相关类，包括了水平分片和垂直分片。

GDD 的主要信息列表如下：

数据库：

```
public class Database
{
        public string name; //数据库名称
        private List<Table> tables;   //数据库包含的表
}
```

表：

```
public class AbstractTable
{
        public string name;//表的名称
        private List<Field> fields;//表中所包含的字段
        public Database db;//  该表所属的数据库
        public int fragmentType;//该表的分片类型
}
public class Table:AbstractTable
{
        //包含的水平分片
        private Dictionary<string, HFragment> hfHt;
        //包含的垂直分片
        private Dictionary<string, VFragment> vfHt;
```

```
}
```
字段：
```
public class Field
{
        public AbstractTable table;//该字段所属的表名
        public string name;//字段名称
        public string type;//字段类型
        public bool key;//是否是主键
}
```
站点
```
public class Site
{
            public string name;//站点名称
            public string host;//站点 IP
            public int port;//站点端口
            public int state;//站点状态
            public bool IsControlSite;//是否是控制站点
}
```
表的分片信息的相关类列举如下：

表的分片信息
```
public class Fragment<T>
{
        public string name;//表的分片的名称
        public Site site;//该分片的表所属的站点
        private List<T> fragmentParts;//分片所含的谓词信息
}
```
水平分片，按谓词分类
```
 public class HFragment : Fragment<Predicate>
{
        public HFragment(string name, Site site)
            : base(name,site)
        {}
 }
```
垂直分片，按所包含字段分类

```
public class VFragment:Fragment<Field>
{
        public VFragment(string name, Site site):base(name,site)
        {}
 }
```

分片表

```
public class FragmentTable : AbstractTable
{
        //水平分片
        public HFragment hf;
        //垂直分片
        public VFragment vf;
       //在哪个站点上
        public Site site;
}
```

谓词类

```
public class Predicate
{
        public Field field;//左值，肯定是一个字段
        public int Operator;//操作符
        public object operand;//右值
}
```

我们会在程序的初始化的时候在控制站点中导入 GDD 的相关信息，所有关于 GDD 的原始分片、站点信息都被存放在一个文本文件中。

4.6 语法分析器

使用了 C#的正则表达式功能对 SQL 语句进行分析和提取。正则表达式是搜索、替换和解析复杂字符模式的一种强大而标准的方法。正则表达式语法较之普通代码相对麻烦一些，但是却可以得到更可读的结果，与用一长串字符串函数的解决方案相比要好很多。

我们设计的正则类的关键定义如下：

```
//select语句
public static String SELECT_SQL = SELECT + SELECT_PART + FROM + FROM_PART + "(" + WHERE
+ WHERE_PART + ")?";
```

```
//insert语句
public static String INSERT_SQL = "insert\\s+into\\s+" + TABLE+ "\\(" + FIELD_PART +
"\\)\\s*values\\s*\\(" + OPERAND_PART + "\\)";
//delete语句
public static String DELETE_SQL = "delete\\s+from\\s+" + TABLE + "\\s+where\\s+" +
WHERE_PART;
//……
```

并通过诸如以下的函数进行分析

```
Regex pt = new Regex(SQLRegex.SELECT_SQL, RegexOptions.IgnoreCase);  /*忽略大小写*/
MatchCollection mt = pt.Matches(sqlStr);     /*返回匹配的match对象的引用*/
```

从而得到输入的 SQL 语句中的各部分关键成分的字符串。

4.7 查询优化

查询树

有关查询树的相关工作主要包括两个方面，一是查询树的生成，二是查询树的优化。

查询树的生成

在生成查询树之前需要先定义查询树的相关的类。分别列举如下：

节点的抽象类的定义

所有查询树中的节点的定义均从该节点继承

```
public abstract class Node
{
   public Node parent;//  父节点
   public int operationType;//操作类型
}
```

Union 节点

```
public class UnionNode:Node
{
private List<Node> nodes;//Union 中所包含的所有的子节点
}
```

Project 节点

```
public class ProjectNode:Node
{
   private List<Field> fields;//所需要映射的字段列表
   public Node child;//Project 的子节点
```

```
}
```

Select 节点

```
public class SelectNode:Node
{
   public Predicate predicate;//Select 所包含的谓词
   public Node child;//Select 的子节点
}
```

Join 节点

```
public class JoinNode:Node
{
      public Node leftChild;//join 节点的左孩子
      public Node rightChild;//join 节点的右孩子
      public Predicate predicate;//join 的连接谓词
}
```

叶子节点

```
public class LeafNode : Node
{
   public Table table;//该叶子节点所包含的表
   public Node child;//该叶子节点的子节点
}
```

叶子节点的孩子的分片节点

```
public class FragmentNode:Node
{
   public FragmentTable fragmentTable;//分片的表
   public Node child;//其孩子为 null
}
```

定义了这些节点之后，再根据 parser 的结果匹配 SQL 查询语句中的实现定义的格式，就可以得到原始的包含分片信息的树。每个 join 之前都会现有 project 的操作，以减少运算量。

查询树的优化

在原始查询树的基础之上，我们所做的查询树的优化主要包括以下几个部分

1.水平分片优化（fragmentOptimizeHorizontal ()）

主要过程是：首先将和谓词有冲突的水平分片进行删除并剪枝，然后将 union

节点上移，选择节点下移。

2. 垂直分片优化（fragmentOptimizeVertical()）

主要过程是：以投影所需要的字段作为依据，判断是否有一些垂直分片没有包含投影所需的任何字段，如果是这样的话，就可以将这个垂直分片删掉并剪枝。

3. union 运算上移（fragmentLiftUnion()）

为了减少运算量，最后还需要将 union 运算上移并且将 project 运算下移。

基本的方法和老师上过课讲的内容是一致的。

4.确定每个节点的执行站点

这个优化是我们所欠缺的，程序中默认为每个非叶子节点的执行站点都是在其最左端的孩子的节点，如果需要考虑这个的优化的话，我们认为可以根据分片信息的先验知识实现指导每个叶子节点的数据量，然后再层层递归，预估得到每个节点传输的数据量，根据数据量的大小来决定是从站点 A 发到站点 B 执行还是从站点 B 发到站点 A 执行。

5.任务分工

这次的大作业是在大家共同讨论，通力合作之下完成的，尤其是优化那一块，结合我们自己的理解，每个人都提出了一些看法，最后汇总才得到我们的最终程序。具体的执行过程中的分工如下所示：

黄科满（队长）：整体设计和框架构建，数据站点管理，底层数据库操作，控制端操作，通讯，进度掌控

来洪波：GDD 的设计，查询树的生成和优化

辛乐：客户端的设计，语法分析器，GDD 的部分设计

6.总结

6.1 黄科满总结

写到这，一学期的分布式数据库也基本上到了尾声。回想开学初很多朋友跟我说这门课是一门杀手级别的课程，课程大作业特别有分量，但是终究还是坚持了下来，完成了这门课程。不能说对这门课程学的多好多深，但是通过本次课程的训练，对于对数据库基本零基础、基本未接触过有规模系统构建的我而言，收获颇丰。主要在于以下的几个方面：

1、 通过一学期的课程学习，从原理上充分的认识和学习了数据库的内容。感谢

周老师和冯老师的深入的讲解让我对分布式数据库有了充分的认识，为我们完成整个系统提供了大量的知识基础。

2、 在完成大系统的过程当中，进度的掌控和团队成员的交流相当重要，这方面还不太成熟，最终还是需要最后几天的连轴转。事实上，在进度的安排上需要在完成系统分析设计的基础上进行有效的设计，并且需要充分的考虑各种突发事件，例如说考试、其它课程 Project 等因素，为每部分留出足够的时间余量。团队成员的交流对于整个进度的推动有着重要的作用。每个人往往会有自己的思维死角，有时候容易钻牛角，而这时候成员之间的交流则能够给彼此提供新的思路形成新的解决方案。例如在做数据分包的时候最开始我采用的是独个记录处理的方式，处理起来相当的复杂。后来在讨论当中，辛乐提出将其交给本地数据库来进行的思路，最后讨论形成数据分包的新的处理逻辑，使得数据分包部分顺利的完成。

3、 对于一个大系统而言，调试的工作相当重要，而且也相当的耗时。在整个系统的设计之初，我们就有效的实现系统功能的模块化，每个成员可以对自己部分模块完成以后进行调试以确保模块的正确性。例如在我负责的这些部分，我都为每个模块编写了测试命令以及测试方案，确保在整合过程中这些模块功能的正确性，降低整合系统所花的时间。事实上在整合的过程中，这些模块也争气的基本上没有出问题。另外就是调试是一个相当重要的工作，在调试的过程中往往能够发现系统中可能存在的 bug。例如说在进行数据导入的过程当中，我们采用的是数据库提供的命令 Load Data。但是在测试的过程当中却出现了以下的问题：

1） 当出现数据量过大的时候会导致内部错误的出现。为此我们需要在出现异常的时候采用新的方式来进行导入。在本系统中采用的则是对数据进行分批插入，将数据分成几个文件，然后再利用 Load Data 进行导入

2） 使用 Load Data 导入数据的时候，由于对数据格式有这比较严格的要求，最开始我们直接导入以后会发现在分片的时候无法获得正确的数据。最开始以为是系统的问题，直到后来才直到原来是数据格式有问题，最后我自己编写了程序对原始数据格式进行调整，才能够顺利的完成整个数据的导入和分片。

正因为在调试的过程当中可能出现自己事先没有预想到的问题，因此调试工作还是需要预留较长的时间的。

4、 系统的整体设计相当的重要，特别是不同功能模块之间的松耦合，对于系统的搭建和整合有着重要的意义。在设计系统的过程当中，控制端负责对命令进行解析，并且生成命令执行计划，然后各个数据站点执行命令计划，使用

命令计划实现控制端和数据站点的分离，使得数据站点的调试和控制端的调试可以分开进行，降低在调试中的复杂度。使用虚拟包 Packet 类，则将执行逻辑与网络通讯部分有效的隔离，使得通讯部分成为独立的模块。事实上这部分也是最先完成的。

最后我们构建的系统并不太理想，尽管最终的执行结果基本上都是正确的，但是执行时间却太长，仔细的分析原因，主要有以下的几个方面：

1、 将 Join 操作直接交给了数据库来进行。在执行的过程中充分的体会到了 Join 操作的时间复杂度，基本上在执行每个查询命令的过程当中需要等待的都在于 Join 命令。为了降低执行的时间，有必要自己编写一套方法来优化 Join 操作。

2、 其次是在优化生成执行命令计划的过程当中，没有最终实现完整的优化，在进行 Join 操作之前没有最大限度的降低 Join 操作的数据量，这也是时间比较长的原因。

总之，经过我们团队的努力我们顺利的完成了课程的学习以及分布式数据库的搭建，再次感谢周老师和冯老师的讲解，感谢范举助教和检查师兄师姐的帮助。还有感谢团队成员来洪波和辛乐的积极努力，忘不了我们一起趴在电脑前独行的分析异常发生的原因，忘不了讨论过程中的集思广益，正是你们的努力让我们顺利的完成了本次的大作业。

6.2 来洪波总结

一个学期的分布式数据库课程就在连续几个晚上的熬夜和最后的顺利验收中基本划上了尾声。对于数据库零基础的我来说，选这门课之前还是做好了充分的困难准备的，好在上课的时候无论是周老师还是冯老师都讲的深入浅出，所以听起来倒并没有觉得很累。说到收获，主要可以从两个层面进行展开，分别是技术层面和组织层面。

作为完全没有数据库基础的我来说，要亲手搭建这样一个平台确实是非常的不容易的，所以技术方面最大的难题就是理论知识的匮乏和实际编程经验的欠缺，以至于在最初的几周曾经一度怀疑是不是能够把这次大作业顺利扛下来，不过在和老师、助教数次的交流和探讨之后，终于对于系统的整个框架、模块和原理有了一个比较明确的认识，也最终选定了我们小组的开发环境的 C#加 MySQL，之前虽然有 C 和 C++的基础，但是 C#毕竟还是有少许的不同，MySQL 就更不用说了，以前根本就没有用过。于是乎，从第六周开始，开始翻阅工具书，了解常用的 C#函数的写法，MySQL 的相关命令，存储方式等等，并且在中期前对于这两

个软件的使用心中终于有了一点底，但是相对于那些中期就已经有可以运行的程序框架的小组来说，我们的进度还是落后很多，因为我们小组的工作才刚刚展开。

在有了前期的铺垫之后，中期之后的工作就进行的顺利了很多，具体就我而言，我负责的是树的生成和优化，树的生成这一块其实不是很难，但是很繁琐，因为需要定义各种不同的节点类以及数据库的各种关键词映射到 C#的响应的类。因为之前也比较缺乏经验，所以说一开始设计的类并不够用，而且很多时候还缺少大量的功能函数，这就需要在后来生成树的时候不断的去完善前面的类的设计，确实是一项既消耗脑力又消耗体力的过程，不过还好的是终于在 14 周之前能把原始树打印出来了，甚为不易。后来有花了一周时间把树变成了一系列 SQL 语句的命令集合，倒也没有太多的技巧，但是我觉得将四个站点的要执行的所有命令先全部生成，然后将其批量发给四个站点让他们去各自执行的创意还是蛮不错的，这样就将有关树的程序和与底层数据库连接的程序完全隔离了，很好的实现了程序的模块化。树的优化就是在验收前一周熬夜做的，就生成树主要做了 selection 的下移、union 的上移、join 的重新分配以及剪枝等。优化的方法和老师在期中考前讲的内容比较一致，不过最后验收的时候 union 的上移并没有完全实现，在一定程度上影响了时间。无论如何，能做成这样对于我们组来说已经是非常不错的了。

另一方面，我觉得也很有必要探讨一下组织层面的东西，和这次的大作业相比，以往我们编过的程序就显得有点小打小闹了，他不是一个仅仅通过 deadline 前的一两天的熬夜就能完成的，而是合理的计划安排而循序渐进的。所谓的组织层面的内容，无非就是两方面，人员的分配和进度的控制。根据往年小组的项目经历，我们三人也进行了组内的分工，黄科满作为队长对于程序的各个模块进行了大致的划分并且主要负责其中的通讯模块和数据库的底层操作模块，辛乐负责界面、parser 模块和一部分的 GDD，而我则负责 GDD 的整合，树的生成和优化模块，分工的好处是可以尽量减小工作的重叠性，实现时间利用的最大化。进度的安排同样是很有必要的，虽然我们最终的进度比我们预期的进度还是慢了一些，但是这依然不能否认进度控制给我们项目的进展带来的帮助。如果没有之前进度控制，即便最后三天连轴转，估计也无法完成整个项目。所以说，只有一个好的团队才能做出一个合格的程序。那么什么是好的团队？好的团队=合理的人员分工+严格的进度控制。

在整个项目的进展过程中，还需要感谢很多人，首先当然是周老师、冯老师和范举助教，一遍一遍的向我们解释一些或许在他们看来很简单的问题。还有就是我们组的黄科满和辛乐，作为队长，黄科满定期的组织我们讨论开会，对于进度严格把关；而作为唯一的女生，辛乐不仅在程序上完成的很出色，设计了很漂

亮的界面，在我们一起熬夜的时候她也是最好的后勤。

当然了，遗憾还是存在的，具体到我的树这一块的话，最终并没有考虑传输量的优化，所以最终的优化还是不完全的，最后还是没时间作这一块了，可以说是优化过程中最大的遗憾。

6.3 辛乐总结

作为一个非计算机专业的学生，我在上课的过程中的情绪经历了忐忑-怀疑-抓狂-欣喜的波澜壮阔。从选课时师兄师姐的口口相传，到开始上课时周老师的谆谆教诲，我都能体会到这门课的重量级别。此外，之前没接触过数据库知识、没有软件系统的开发经历对于我来说也是较大的困难。但是为了在有限的学分内最大程度地提升自己的能力以尽快适应今后的项目要求，我还是毅然选了分布式数据库。在惴惴中开始了 ddb 的求学之路，我也时刻告诫自己，既然基础薄弱，就要认真听课勤于探索，并尽量提前学习大作业的预备知识。

在课堂学习中，周老师、冯老师和助教范举师兄给了我很大的启发，让我收获到了关于数据库及分布式数据库的基础知识，并引起了我很大的学习钻研的兴趣，理论上的学习让我深刻领会到分布式数据库系统的要求之严谨，也愈发对自己能否完成大作业产生了怀疑，不知道和两位同学一同协作能否完成这次艰巨的任务。硬着头皮从零基础做起，我们开始了大作业的实施。一开始明确了分工，大家统一了设计思路，便分头去学习具体的编程实现方式。我们选用了 C#和 Mysql 来实现，应该说对现阶段我们的水平来说是较为合适的。随着编程的深入，我最大的感触就是大系统和若干个小程序之和二者之间的差异太显著了，即使这一点在各种软件类的基础课上一再被提及。由于之前学习过 C 和 C++，也编写过一些一个人做一两天就可以完成的小程序段，但这次的代码量使得我一度对程序的框架很迷茫，当时心里非常恐慌，感觉这样的思维框架和方式自己难以胜任。好在来洪波和黄科满两位同学思路非常清晰，在和他们的讨论过程中，我逐渐明确了框架和系统的定义，确定了自己的任务和方向。直到最后项目验收时，我们一条条指令都顺利地运行完毕，得到了正确的结果返回值，我的心情非常雀跃，感受到了分布式数据库的魅力所在，感受到了团队协作的愉悦，感受到了经历了磨砺之后的获益匪浅。

不得不说这是一段美好的共同奋斗的回忆，大家一起热烈讨论甚至争论方案的时候、全心全力查错调试的时候、各自关心进展直至最后整合的时候，各种通力合作的情景都让我记忆犹新。最大的感触是技术和组织需要协调发展，组织架构是技术协作的基础平台支持。对于我们来说，三个人之间的沟通和交流基本顺

畅。总结来说，我认为在合作中应当注意以下的关键词：

首先是勤勉。在基础的分工之后，我们明确了努力的方向，黄科满作为队长对于程序的各个模块进行了大致的划分，并且主要负责其中的通讯模块和数据库的底层操作模块，来洪波负责 GDD 的整合，树的生成和优化模块，我负责界面、parser 模块和一部分的 GDD。大家都努力的做好自己的事情，特别是他们两位同学，效率很高，在需要的时候给我以帮助和鼓励，这样的氛围下大家的进展都较顺利，并且很少出现互相等待的死锁现象。黄科满做的通讯和底层数据库操作的部分是我们项目的基础，来洪波定义的 GDD 给我们提供了各种类的定义和实现方法。在大家的努力下，项目的实现便水到渠成了。

其次是沟通，在写代码的过程中，往往方法比努力重要的多，有时复杂的实现换个角度便会找到简便易行的实施策略，这就是 idea 的重要性。在组织的协作沟通中，火花会时常闪现，比如说关于数据导入方面，在具体实现时，本来我们的想法是逐条判断插入，某天我在学习数据库基础操作时，突然想到可以用 mysql 的功能实现自动判断、建表，在此基础上通过网络进行发送，这样的效率会高很多，而且错误率、代码复杂度会大大降低。诸如此类的一些想法往往让我们很振奋，而大家互相讨论也有助于错误的发现，所以说，这样的合作中，沟通所发挥的效益有目共睹。

最后是控制，所谓管理常常被称为计划、组织、协调、控制的结合，这次的合作中，进度和版本是我们两大问题。关于进度，虽然在队长的努力下基本做到了按时验收、每周讨论，但客观上由于其他考试、大作业的冲击，我们的精力还是被分散了一些，另外主观上可能有一些盲目的乐观，没有能做到完全遵照预期安排，最后还是熬夜了一晚上进行联调；关于版本，我们没有使用成熟的软件进行控制，这个问题在实现的最后很让我们头疼。每个人都是在自己的源程序上进行的修改，整合起来的时候若有问题也习惯用相对独立的小段程序进行调试，这就造成了互相来回发代码的窘态，教训就是应当使用目前成熟的软件进行代码管理。

最后再次感谢同组的两位同学的努力和帮助，感谢老师和助教的指导，这次的课程以及大作业对我来说将是非常难忘和获益匪浅的经历。