浙江大学

大规模信息系统构建技术导论

分布式 MiniSQL 系统报告

指导老师：鲍凌峰

2022 学年 **春** 学期

组员信息（第一行请写组长信息）

学号	姓名
3190102994	陈云奇
3190103296	严昕辰
3190104538	朱亦陈
3190103313	钱星屹
3190105590	王晨璐

2021 年 5 月 25 日

目录

1 引言.....3
1.1 系统目标.....3
1.2 设计说明.....3
2 系统设计与实现.....5
2.1 系统总体架构.....5
2.2 Client 设计.....5
2.2.1 架构设计.....5
2.2.2 工作流程.....6
2.3 Master Server 设计.....8
2.3.1 架构设计.....8
2.3.2 Master Server 内部通信设计.....9
2.3.3 Master Server 整体工作流程.....11
2.4 Region Server 设计.....12
2.4.1 架构设计.....12
2.4.2 工作流程.....13
3 核心功能模块设计与实现.....14
3.1 Socket 通信架构与通信协议.....14
3.1.1 系统通信架构.....14
3.1.2 通信协议设计.....14
3.2 分布式查询与跨机 Join.....16
3.3 数据分布与集群管理.....16
3.3.1 服务发现.....17
3.3.2 服务注册.....17
3.4 负载均衡&数据备份.....17
3.4.1 Create Table.....18
3.4.2 Other Operation.....18
3.5 容错容灾.....18
4 系统测试.....19
4.1 初始化.....19
4.2 创建表.....19
4.3 插入记录.....19
4.4 查询记录.....20
4.4.1 条件查询.....20
4.4.2 全部查询.....20
4.5 删除记录.....20
4.6 删除表.....21
4.7 join 操作.....21
4.8 负载均衡.....21
4.9 副本管理与容错容灾.....22
5. 总结.....22

1 引言

1.1 系统目标

本项目是《大规模信息系统构建技术导论》的课程项目，在大二春夏学期学习的《数据库系统》课程的基础上结合《大规模信息系统构建技术导论》所学知识实现的一个分布式关系型简易数据库系统。

该系统包含 etcd 集群、客户端、主节点、从节点等多个模块，可以实现对简单 SQL 语句的处理解析和分布式数据库的功能，并具有数据分区、负载均衡、副本管理、容错容灾、sql 语句的 join 实现等功能。

本系统使用 Go 语言开发，并使用 Github 进行版本管理和协作开发，由小组内的五名成员共同完成，每个人都有自己的突出贡献。

1.2 设计说明

本系统由小组 5 位成员合作编写完成，使用 Go 开发，GoLand 作为集成开发环境，Github 作为版本管理和协作开发工具，每个组员都完成了目标任务，具体分工安排如下：

成员姓名	学号	分工职责
陈云奇	3190102994	组长。项目整体的架构设计和搭建；基于 etcd 的主节点的服务发现和从节点的服务注册功能；参与主节点和从节点通信协议模块的设计；系统测试
严昕辰	3190103296	客户端 Client 的开发
朱亦陈	3190104538	主节点 Master 的开发，容错容灾、副本管理和负载均衡的功能开发
钱星屹	3190103313	从节点 Region 的开发，建立在 sqlite 基础上，包括 sql 语句的查询、执行；对于每个表的操作记录

		的备份还原
王晨璐	3190105590	负责 Master 与 Client 及 Region 的通讯功能实现，并参与系统测试

表 1 成员分工表

2 系统设计与实现

2.1 系统总体架构

本系统的总体架构设计如下图所示：

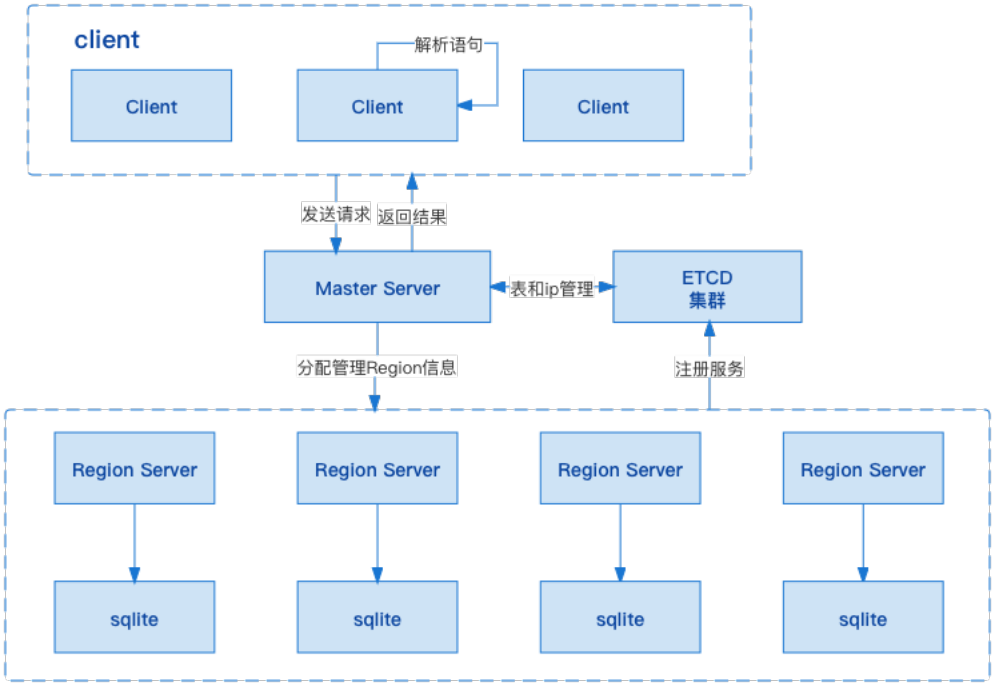


图 1 总体架构图

本系统将整个项目划分为三个模块，分别是 Client，Master Server 和 Region Server，分别对应分布式数据库系统的客户端、主节点和从节点，客户端负责和用户交互，主节点作为通讯中心和调度中心，从节点作为数据库的执行单元，同时主节点和从节点通过 ETCD 集群，对数据表的信息进行统一的管理。

2.2 Client 设计

2.2.1 架构设计

本系统中的 Client 设计如图 2 所示：

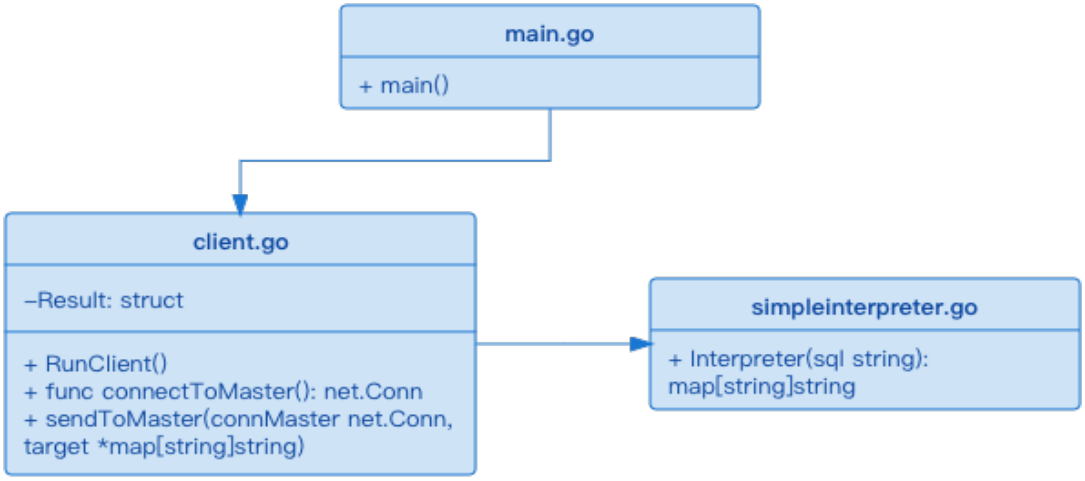


图 2 Client 设计图

Client 通过运行 main.go 调用其他文件，实现全部功能，其中：

- client.go 中定义了 client 启动函数 RunClient()，实现与用户的信息交互过程，其中调用了 connectToMaster()与主节点建立 tcp 连接，sendToMaster()则在连接的基础上向主节点发送请求。
- simpleinterpreter.go 则是一个简单的解析器,将 sql 语句解析为 map，其中 kind 字段记录了具体操作，如 create、select、delete 等，join 字段记录了该操作是否为 join(方便 Master 的特殊处理)，name 字段记录了操作涉及的表单名字，sql 记录了用户输入的 sql 语句……

2.2.2 工作流程

客户端的工作流程可以用下面的流程图来表示：

图 3 Client 工作流程图

2.3 Master Server 设计

2.3.1 架构设计

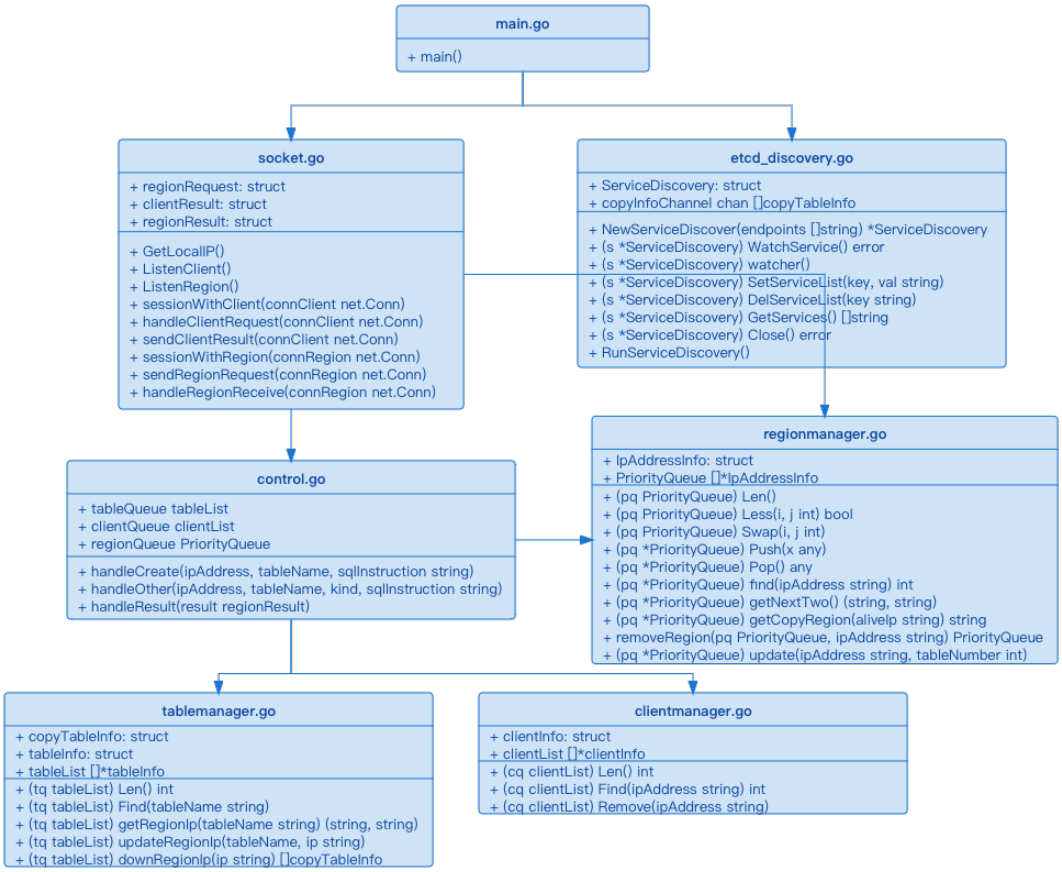


图 4 Master 设计图

MasterServer 的主要职责如下：

- 作为**通讯中心**，负责接收消息，完成消息格式转换及信息提取，同时完成消息的转发；
- 作为**调度中心**，负责完成 Table 的备份管理，容错容灾处理等并发事件调整；

具体架构如下：

- socket.go： Master Server 的消息控制中心，负责监听 Client Request，并根据请求向对应的 Region Server 转发对应格式的 Region Request；同时接

受 Region Server 返回的处理结果，并将其进行数据格式统一之后作为 Client Request 的 Result 返回给对应的 Client；

- etcd_discovery.go：etcd 集群的控制中心，负责集群成员的注册、服务以及退出管理；
- control.go：作为 Master Server 处理消息的 Toolkit，负责针对不同的请求以及处理结果做出对应的消息转发或中间处理；
- lientmanager.go：记录与 Master Server 连接的 Client 的信息，包括 IP 等关键信息
- regionmanager.go：记录与 Master Server 连接的 Region Server 的相关信息，包括 IP、所存储的 Table 数量等关键信息；同时实现将 Region 按照所存 Table 数量排序 并存储为一个 Heap，以方便完成负载均衡以及数据备份；
- tablemanager.go：记录分布式数据库中所存储的所有可用 Table 的相关信息，包括表名、存储该表的 Region Server 的 IP 等；同时提供了维护该 Table List 所需的接口函数；

2.3.2 Master Server 内部通信设计

首先，需要说明的是： Client、Master Server 以及 Region Server 之间的连接是非常稳定的 TCP 连接，其体现在 Go 语言中 就是一个 net.Conn 的数据类型；为了实现并发处理，在 Master Server 中为每一个连接单独设置了一个 goroutine 来对其进行处理；而 Master Server 的一大主要功能就是对请求以及消息的转发，这就要求 Master Server 在其内部有一套复杂的 goroutine 见的通信策略以保证消息转发的通畅，这一部分将对该策略进行详细的介绍。

在本系统中，使用 Go 语言中的 channel 这一类型的数据进行 goroutine 间通信的管理。 图 2 中可见，在 clientmanager.go 以及 regionmanager.go 中，定义针对与 Master 连接的 client、region server 的信息管理结构 clientinfo 和 regioninfo，在该结构体内部加入 channel，即可实现 channel 与连接的绑定。

```
type clientInfo struct {
	ipAddress string
	resultQueue chan middleResult
}

type IpAddressInfo struct {
	ipAddress          string
	tableNumber        int
	requestQueue       chan regionRequest
	copyRequestQueue       chan string
	copyInfoQueue           chan string
	index                 int
}
```

对于 Client 发送来的请求，通过 channel 的 Master Server 的内部消息流程如下：

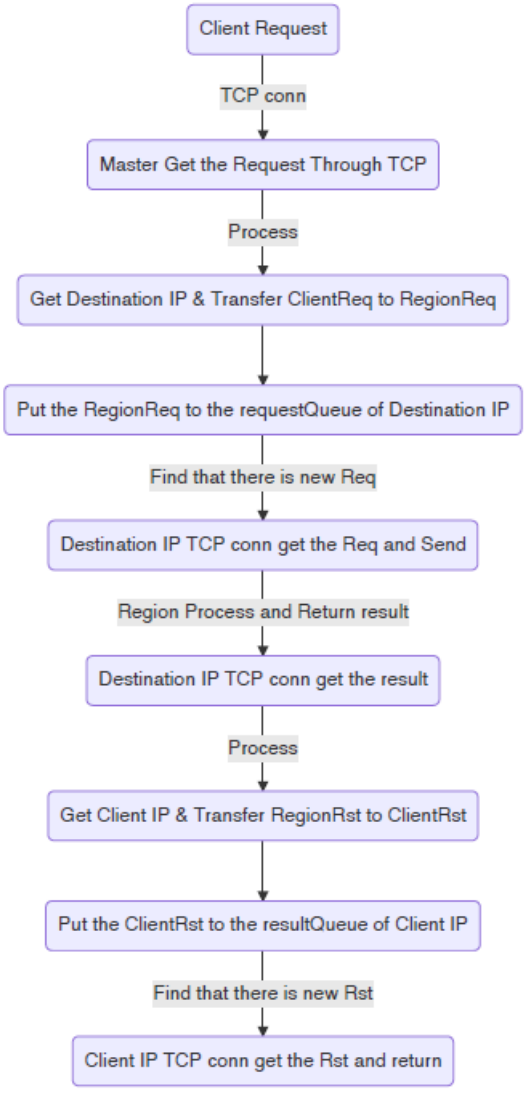


图 5 Master 内部的 channel 流程图

2.3.3 Master Server 整体工作流程

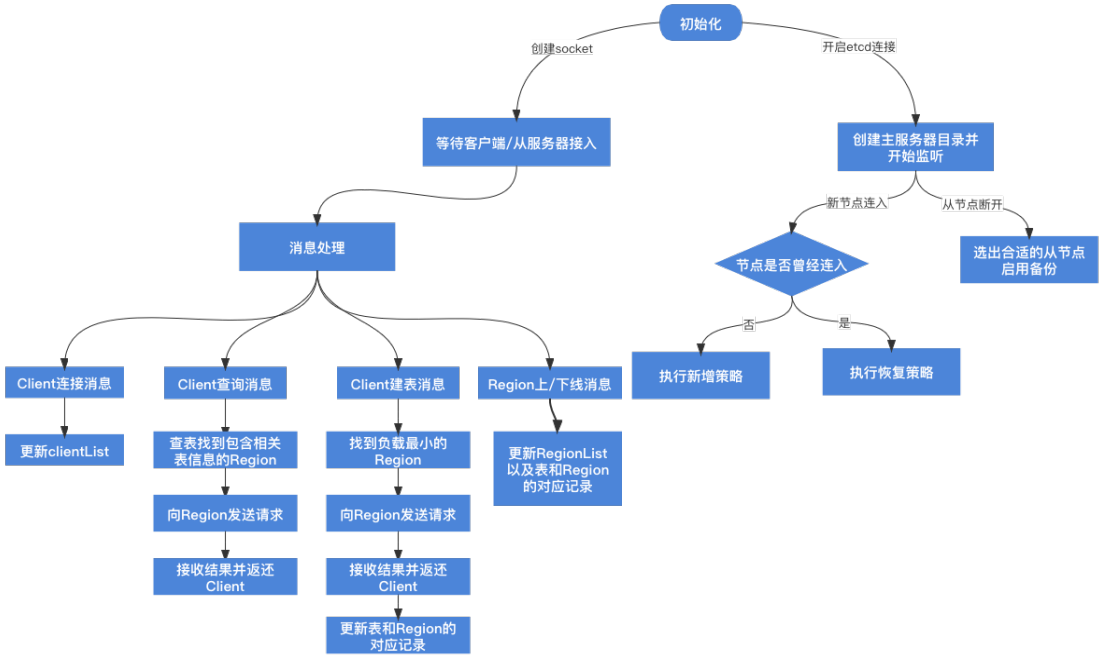


图 6 Master 工作流程图

从服务器管理自己的数据库，并和主服务器通信：

- main.go 中定义了 Region 的主要操作，处理来自 Master 的请求、返回请求处理结果，其中调用了其他文件中的函数实现对本地数据库的操作，同时包含对表信息的 Log 登记，便于从节点宕机时实行备份策略。
- etcd_register.go 负责使用 etcd 进行服务注册，便于主节点监听当前节点，实现集群管理。
- socket.go 给出了与主节点建立连接的函数，负责从节点通讯。
- query.go 负责对本地数据库进行查询、修改等操作。
- backup.go 管理了本地数据库中每个表的操作记录，便于从节点宕机时实行备份策略。
- exec.go 在接受到来自主节点的备份命令后，根据传来的相关表的操作记录，逐条实现操作，完成表的备份。

2.4 Region Server 设计

2.4.1 架构设计

Region Server 设计如下图所示：

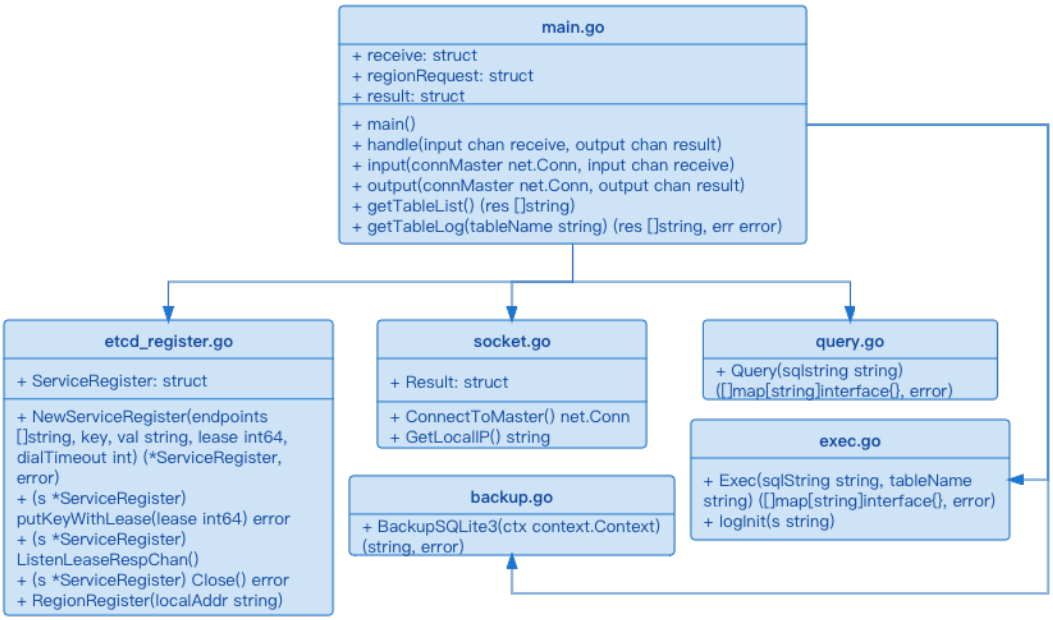


图 7 Region 整体设计图

Region Server 建立在 sqlite 的基础上完成对数据库的相关操作，并添加了 Table 的操作备份和还原策略，同时添加与 Master Server 部分的通信功能：

- main.go 中定义了 Region 的主要操作，处理来自 Master 的请求、返回请求处理结果，其中调用了其他文件中的函数实现对本地数据库的操作，同时包含对表信息的 Log 登记，便于从节点宕机时实行备份策略。
- etcd_register.go 负责使用 etcd 进行服务注册，便于主节点监听当前节点，实现集群管理。
- socket.go 给出了与主节点建立连接的函数，负责从节点通讯。
- query.go 负责对本地数据库进行查询、修改等操作。
- backup.go 管理了本地数据库中每个表的操作记录，便于从节点宕机时实行备份策略。
- exec.go 在接受到来自主节点的备份命令后，根据传来的相关表

的操作记录，逐条实现操作，完成表的备份。

2.4.2 工作流程

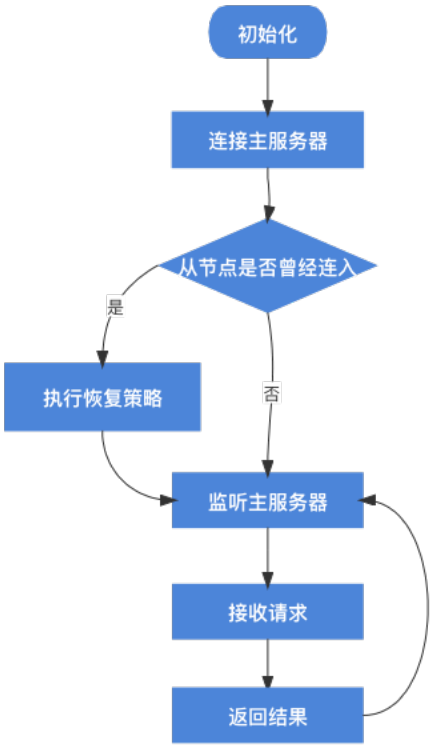


图 8 Region 整体设计图

3 核心功能模块设计与实现

3.1 Socket 通信架构与通信协议

3.1.1 系统通信架构

本系统将 Master 作为通讯管理中心，通过 Socket 连接建立稳定可靠的通信与异步的 IO 流。Client 向 Master 请求服务，Master 接收消息后将任务分配给合适的 Region，并将从 Region 获取的数据信息返还给 Client。系统的总体通信架构如下图所示：

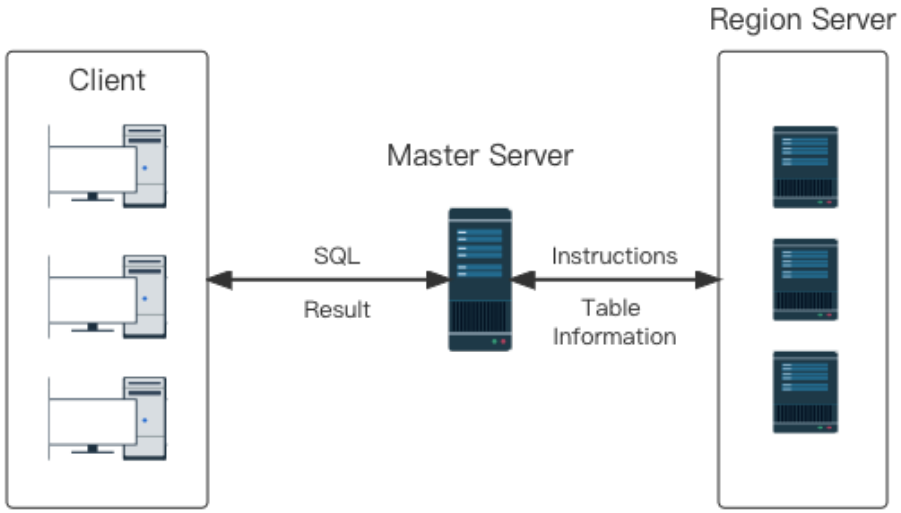


图 9 系统通信架构图

其中 Master 需要一直保持在线，然后 Region 和 Client 依次与 Master 建立连接，Master 与 Region 的连接是一直存在的，除非 Region 意外宕机，Client 和 Master 的连接则是可以断开的。不同模块之间的通信基于 tcp 协议，都是全双工的。

3.1.2 通信协议设计

其中 Master 和 Region、Client 的通讯消息都是以 json 格式定义的，传输前将结构转为字节流，以字节流方式传输，在抵达终点后对其复原。

```
// Master 返回给 Client 结果
type clientResult struct {
```

```
	Error string                            `json:"error"`
	Data   []map[string]interface{} `json:"data"`
}

// Master 向 Region 发送的请求
type regionRequest struct {
	TableName string
	IpAddress string
	Kind         string
	Sql          string
	File         []string // 用于 copy table
}

// Region 给 Master 返回结果
type result struct {
	Error       string                          `json:"error"`
	Data         []map[string]interface{} `json:"data"`
	TableList []string                        `json:"tableList"`
	Message    string                          `json:"message"`
	ClientIP   string                          `json:"clientIP"`
	File         []string                        `json:"file"`
	TableName string                        `json:"tableName"`
}
```

主节点作为通讯中心，和 Client 与 Region 共同维护特定的通讯协议。

Master 向 Region 发送的请求包含以下部分：

1. 待操作表名
2. 发起请求的客户端 ip 地址
3. 操作类型（查询、复制、重建等）
4. 原始 sql 语句
5. File 字段，该字段在备份策略时生效，即当某从节点宕机后，主节点找到与宕机从节点储存了同样表的其他从节点，获取遗失表的操作记录文件，并发送给重新选定的从节点。

Region 向 Master 返回的结果包含以下部分：

1. 错误信息
2. 查询数据结果
3. 该从节点包含的所有 Table 信息

4. 返回信息类型
5. 发起请求的客户端 ip
6. 当前请求的操作涉及的表名
7. 还有用于执行备份策略的 File 字段。

Master 向 Client 返回的结果则包含

1. 错误信息
2. 请求返回的具体数据，供给 Client 输出操作结果。

3.2 分布式查询与跨机 Join

本系统实现了分布式查询，即客户端执行 SQL 语句时，Master Server 会转发给不同的 Region Server 进行操作；同时操作也会可能涉及对不同服务器中的多张表。

我们还实现了跨机 Join 功能，若 join 请求的表位于不同服务器，Master 会找到请求涉及的服务器，将需要的表通过该表的 sql 语句 log 临时复制到同一服务器中，实行 join 操作后再返回结果。

3.3 数据分布与集群管理

本系统作为一个分布式数据库，必须对数据分布进行合理的设计。在本系统中，每个从节点管理一个数据库，对应若干张表。主节点记录了每个从节点和自己管理的表的对应关系。当客户端需要对某张表进行操作时，则向 Master 发送请求，Master 查找到包含该表的所有 Region 进行访问；当有从节点挂掉，主节点执行容错容灾策略时，同时也要更新自己所存储的表的对应关系。

集群管理则通过 etcd 进行。etcd 是开源的、高可用的分布式 key-value 存储系统，可以用于服务的注册和发现。在本项目中我们主要利用 etcd 完成 Master Server 端的服务发现和 Region Server 端的服务注册功能。服务发现要解决的也是分布式系统中最常见的问题之一，即在同一个分布式集群中的进程或服务，要如何才能找到对方并建立连接。本质上来说，服务发现就是想要了解集群中是否有进程在监听 udp 或 tcp 端口，并且通过名字就可以查找和连接。

我们使用 go 语言的 clientv3 相关 API 完成与 etcd 的相关交互操作。

3.3.1 服务发现

服务发现需要实现以下基本功能：

- **服务注册**：同一 service 的所有节点注册到相同目录下，节点启动后将自己的信息注册到所属服务的目录中。
- **健康检查**：服务节点定时进行健康检查。注册到服务目录中的信息设置一个较短的 TTL，运行正常的服务节点每隔一段时间会去更新信息的 TTL ，从而达到健康检查效果。
- **服务发现**：通过服务节点能查询到服务提供外部访问的 IP 和端口号。比如网关代理服务时能够及时的发现服务中新增节点、丢弃不可用的服务节点。

3.3.2 服务注册

每个从节点连入主节点时，首先需要在 etcd 中完成服务注册，通过 etcd 进行集群管理，从而确保当节点增加、失效、恢复时，能够监听到，并作出相应的处理。

根据 etcd 的 v3 API，当启动一个 Region Server 的时候，把 Region Server 的地址写进 etcd，注册服务。同时绑定租约（lease），并以续租约（keep leases alive）的方式检测服务是否正常运行，从而实现健康检查。

3.4 负载均衡&数据备份

在本系统中，负载均衡是指在不同的 Region 中尽量存有数量相近的数据库表，负载均衡的粒度为 Table。而数据备份是通过在两个不同的 Region 均有对同一张表的存储完成的；同时，为了保证分布式数据库的一致性，我在 Master Server 中完全并发地对存有同一张表的两个 Region Server 发送相关消息。在本系统中，我们实现数据备份的粒度为 Table，因此当 Table 被

Create 时，数据备份开始，此时需要在保证负载均衡的情况下选择两个 Region Server 存储该 Table，同时维护 Table 与存储该 Table 的 Region IP 的对应关系；而在之后的数据操作中，仅需要保证同时对两个存储该 Table 的 Region IP 发送请求即可。

3.4.1 Create Table

前文提到，为实现负载均衡，在 regionmanager.go 中维护了一个以 tableNumber 为排序依据的 RegionInfo 堆。因此，为保证负载均衡同时完成数据备份的要求，我们仅需从堆中取出 tableNumber 最小的两个 Region 即可。

3.4.2 Other Operation

由于在 Create Table 时维护了一个存储 Table 与该 Table 所在 Region 对应关系的数组 tableInfoList，因此为实现数据备份同时保证一致性，我们仅需 tableList 中找到该 Table 对应的 Region 并同时发送请求即可。

值得一提的是，除 Create Table 以外的的任何操作，只要至少有一个 Region Server 返回操作成功，我们就认为该指令执行成功，其原因在于本系统中的容错容灾功能会对数据库的一致性作二次保证。

3.5 容错容灾

本系统中 Master Server 负责的容错容灾主要包括两个方面。

一方面，Master Server 通过 ETCD 监测到有从节点宕机，立即从正常工作的从节点中选出最优的从节点，也就是存放表数量最少的从节点，通过 Master Server 维护的该宕机节点所包含的所有表名，再次查表找到包含这些表的其他从节点，并从这些从节点中获取相关表的全部操作记录，发送给选择的最优从节点，被选从节点在接受到操作记录后执行这些操作，Copy 消失的表。

另一方面，当宕机的从节点重启后，Master Server 监测到该事件，执行恢复策略，即要求该节点清空数据库，即完成容错容灾流程。

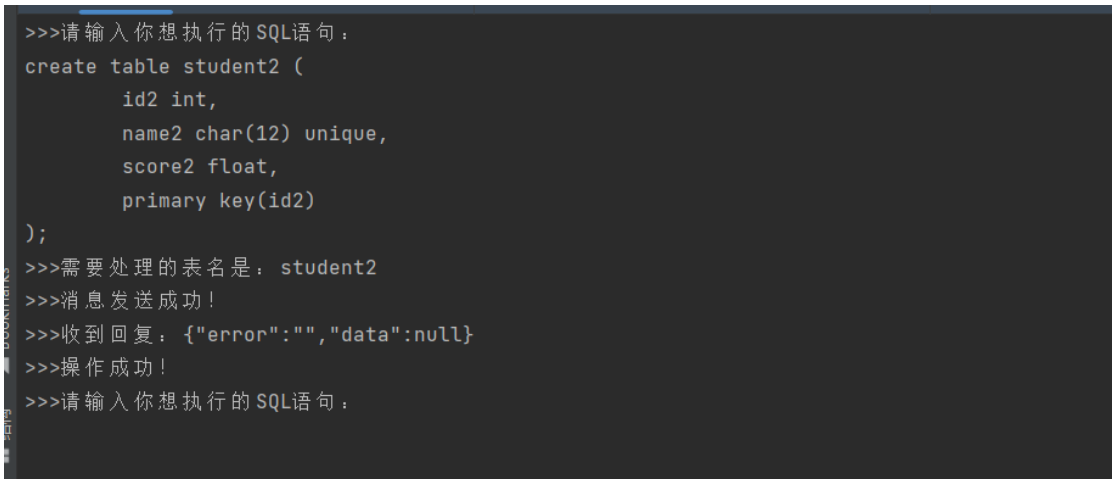
4 系统测试

4.1 初始化

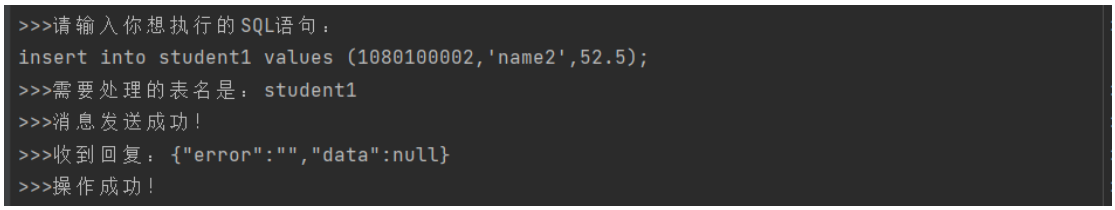
当运行所有准备好的 Master, Region 和 Client 时，整个系统运行初始化的结果是：

- 客户端
- 主节点
- 从节点

4.2 创建表



4.3 插入记录



4.4 查询记录

4.4.1 条件查询

```
select name1 from student1 where score1 >90;
>>>需要处理的表名是：student1
>>>消息发送成功！
>>>收到回复：{"error":"","data":[{"name1":"name1"}]}
>>>操作成功！
>>>查询结果如下：
| name1 |
| name1 |
>>>请输入你想执行的SQL语句：
```

4.4.2 全部查询

```
>>>请输入你想执行的SQL语句：
select * from student1;
>>>需要处理的表名是：student1
>>>消息发送成功！
>>>收到回复：{"error":"","data":[{"id1":1080100001,"name1":"name1","score1":99},{"id1":1080
100002,"name1":"name2","score1":52.5}]}
>>>操作成功！
>>>查询结果如下：
|             id1 | name1 | score1 |
| 1.080100001e+09 | name1 |     99 |
| 1.080100002e+09 | name2 |   52.5 |
>>>请输入你想执行的SQL语句：
```

4.5 删除记录

```
>>>请输入你想执行的SQL语句：
delete from student1 where score1 >90;
>>>需要处理的表名是：student1
>>>消息发送成功！
>>>收到回复：{"error":"","data":null}
>>>操作成功！
>>>请输入你想执行的SQL语句：
```

4.6 删除表

```
>>>操作成功!
>>>请输入你想执行的SQL语句：
drop table stu;
>>>需要处理的表名是：stu
>>>消息发送成功!
>>>收到回复：{"error":"","data":null}
>>>操作成功!
>>>请输入你想执行的SQL语句：
```

4.7 join 操作

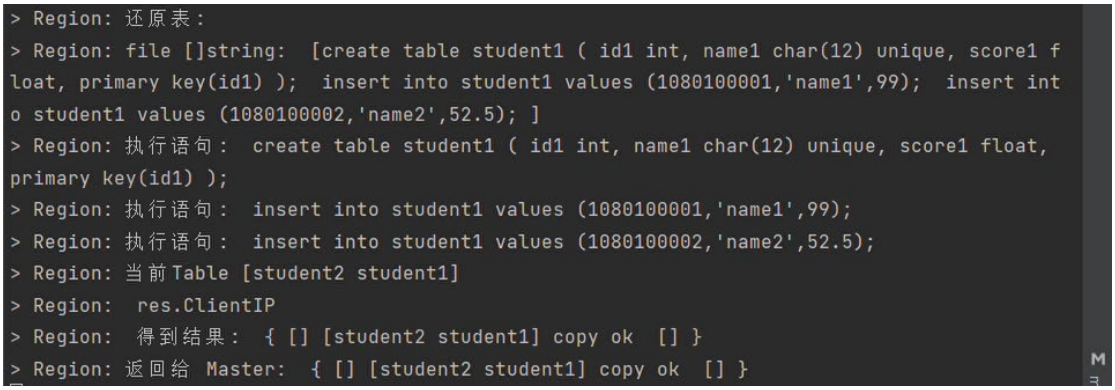
```
select * from student1 join student2;
>>>需要处理的表名是：student1 student2
>>>消息发送成功!
>>>收到回复：{"error":"","data":[{"id1":1080100002,"id2":1080100008,"name1":"name2","name2"
e8","score1":52.5,"score2":73.5},{"id1":1080100002,"id2":1080100009,"name1":"name2","name2"
:"name9","score1":52.5,"score2":79.5}]}
>>>操作成功!
>>>查询结果如下：
|             id1 |             id2 | name1 | name2 | score1 | score2 |
| 1.080100002e+09 | 1.080100008e+09 | name2 | name8 |   52.5 |   73.5 |
| 1.080100002e+09 | 1.080100009e+09 | name2 | name9 |   52.5 |   79.5 |
>>>请输入你想执行的SQL语句：
```

4.8 负载均衡

Master Region 消息记录


```
> Region: rec {create table student3 ( id3 int, name3 char(12) unique, score3 float, primary key(id3) );  2 student3 192.168.119.90:1755 []}
> Region: 执行语句 create table student3 ( id3 int, name3 char(12) unique, score3 float, primary key(id3) );
> Region: 当前Table [student3]
> Region:  res.ClientIP 192.168.119.90:1755
> Region:  得到结果：  { [] [student3] ok 192.168.119.90:1755 [] student3}
> Region: 返回给 Master:  { [] [student3] ok 192.168.119.90:1755 [] student3}
```

4.9 副本管理与容错容灾



5. 总结

本次课程项目由我们小组 5 人合作，完成了分布式关系型数据库系统的搭建，项目由 Go 语言完成，引入 etcd 集群管理，实现了对分布式 sqlite 数据库系统的搭建，完成了分布式存储、负载均衡、副本管理和容错容灾等功能。通过共同设计和搭建我们自己的分布式数据库，我们在实践中加深了对于分布式系统的理解，学习并实践了编程语言 Go，掌握了 etcd 集群管理的方法，受益匪浅。