浙江大学

大规模信息系统构建技术导论

分布式 MiniSQL 系统个人报告

授课教师：	鲍凌峰
学号：	3190102994
学生姓名：	陈云奇
所在专业：	软件工程
所在班级：	软工 1902

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 个人贡献

本系统由小组 5 位成员合作编写完成，使用 Go 语言进行开发，GoLand / VsCode 作为集成开发环境，Github 作为项目控制工具；本人作为小组组长，在开发中主要负责系统整体架构的设计，etcd 服务注册和服务发现的设计开发，参与主节点和从节点通信协议模块的设计以及系统测试

2 系统设计与实现

2.1 系统总体架构

由主导，在和组员讨论后，确定本系统的总体架构设计如下图所示：

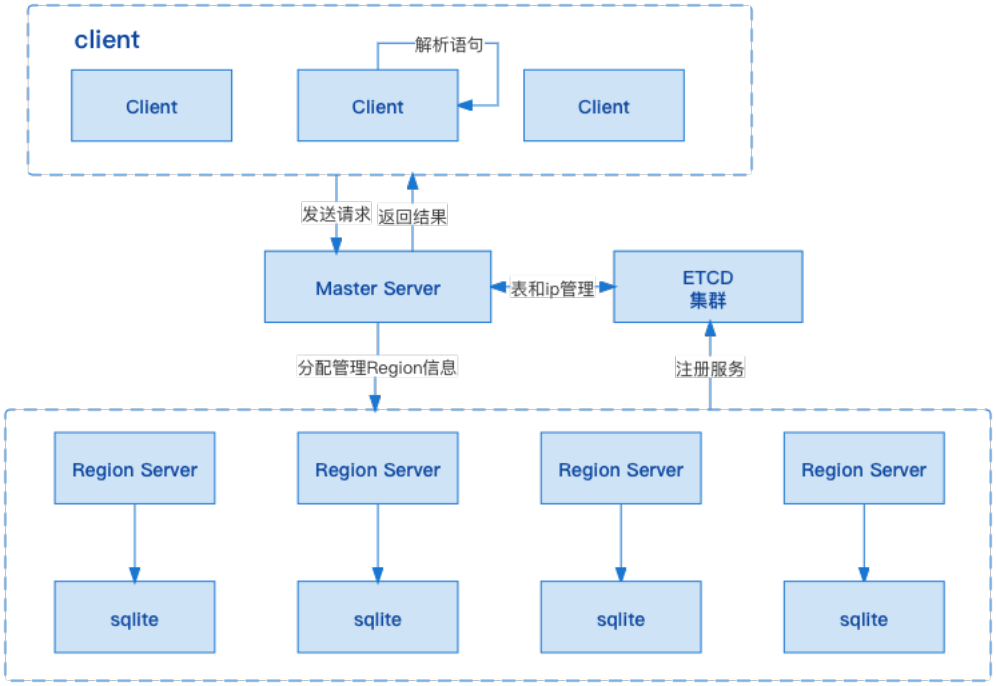

图 1 总体架构图

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，其中主节点作为通讯中心负责收发消息和任务分配，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。

本人在开发中主要负责系统总计架构的设计，etcd 服务注册和服务发现的设计开发，参与主节点和从节点通信协议模块的设计以及系统测试接下来将对我负责的部分进行具体阐述。

2 本人贡献

2.1 系统架构的整体设计

在项目开始编码之前，由本人主导进行项目整体的架构设计，主要工作如下：

- 确定了技术栈为 go 语言，使用 etcd 进行服务注册与发现，使用 sqlite 作为 Region 部分的数据库
- 使用复制 table 的方式完成容错容灾

- 多个 client-单 master-多 Region 的主从架构。

2.2 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。

本人主要负责了集群管理部分的编码实现。

集群管理通过 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

见 etcd_discover.go 功能

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。

```
// 服务发现数据结构
type ServiceDiscovery struct {
    cli        *clientv3.Client
    serverList map[string]string // 服务列表
    lock       sync.RWMutex
}
```

具体函数如下：

```
// NewServiceDiscover 新建发现服务
func NewServiceDiscover(endpoints []string) *ServiceDiscovery
//WatchService 初始化服务列表和监视
func (s *ServiceDiscovery) WatchService()
//watcher 监听 key 的前缀
func (s *ServiceDiscovery) watcher()
//SetServiceList 新增服务地址
func (s *ServiceDiscovery) SetServiceList(key, val string)
//DelServiceList 删除服务地址
func (s *ServiceDiscovery) DelServiceList(key string)
//GetServices 获取服务地址
func (s *ServiceDiscovery) GetServices()
//Close 关闭服务
func (s *ServiceDiscovery)
```

3.3.2 服务注册

见 etcd_register.go

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。

```
// ServiceRegister 服务注册结构
type ServiceRegister struct {
    cli     *clientv3.Client // etcd client
```

```
    leaseID clientv3.LeaseID // 租约 ID
    // 租约 keepalive 相应 chan
    keepAlveChan <-chan *clientv3.LeaseKeepAliveResponse
    key          string // key
    val          string // value
}
```

```
// NewServiceRegister 创建租约注册服务
func NewServiceRegister(endpoints []string, key, val string, lease
int64, dialTimeout int) (*ServiceRegister, error)

// 设置租约
func (s *ServiceRegister) putKeyWithLease(lease int64)

// ListenLeaseRespChan 监听 续租情况
func (s *ServiceRegister) ListenLeaseRespChan()

// Close 注销服务
func (s *ServiceRegister) Close()

// RegionRegister 从节点服务注册功能
func RegionRegister(localAddr string)
```

2.3 协议接口设计

本人参与设计了 Master Server 与 Region Server 的 json 协议接口设计，具体如下。

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 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，项目使用 Go 语言编写，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。本项目是我第一次设计并实现分布式系统，同时也是第一次使用 Go 语言进行大型项目的编写，在此过程中，我充分将课上学到的分布式相关的知识转化为实践，从而较为深刻地理解了相关知识，较为熟练地从零开始掌握了 Go 这一门编程语言，了解到了 etcd 这样的卓越的开源项目，也进一步提升了自己的系统设计能力和团队协作能力，总的来说获益匪浅。