浙江大學

大规模信息系统构建技术导论

分布式数据库个人开发报告

授课教师：	鲍凌峰
姓　　名：	朱亦陈
学　　号：	3190104538
组　　别：	Alien DB
日　　期：	2022.5.25

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、容错容灾等功能。

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进行开发，GoLand / VsCode 作为集成开发环境，Github 作为项目版本控制工具；本人在开发中主要负责 Master Server 的设计与开发，数据备份设计、负载均衡管理以及容错容灾管理等工作，以下文档将对本人工作进行详细说明。

2 系统设计与实现

2.1 系统总体架构

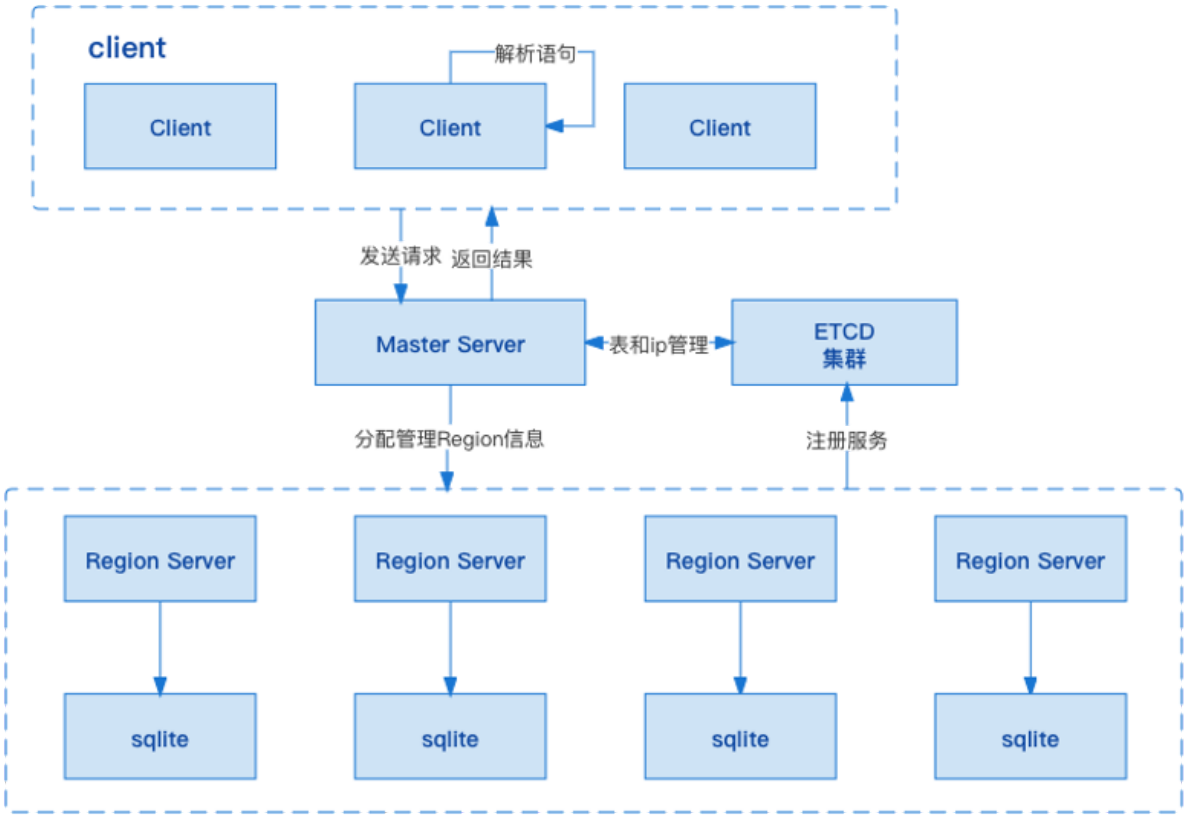


图 1 系统总体架构图

本系统将整个项目划分为3个模块，分别是 Client、Mater 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 Master Server 的职责如下：

- 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度；

2.2 Master Server 设计说明

2.2.1 总体设计

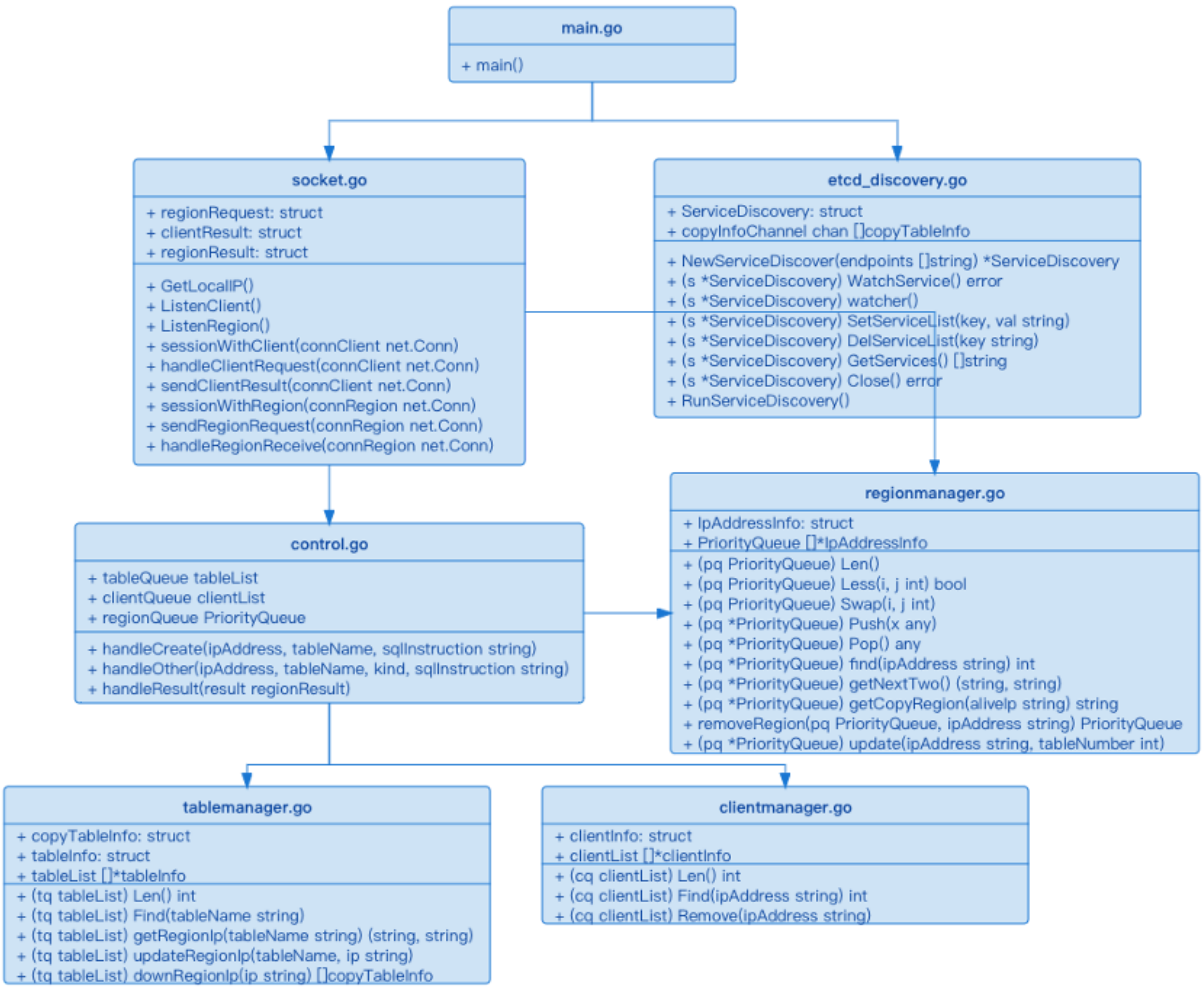


图 2 Master Server 架构图

- **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；
- **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；
- **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；
- **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；
- **tablemaneger.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；
- **clientmanager.go:** 记录与 Master Server 连接的 Client 的信息，包括 IP 等关键信息

2.2.2 Master Server 内部通信设计

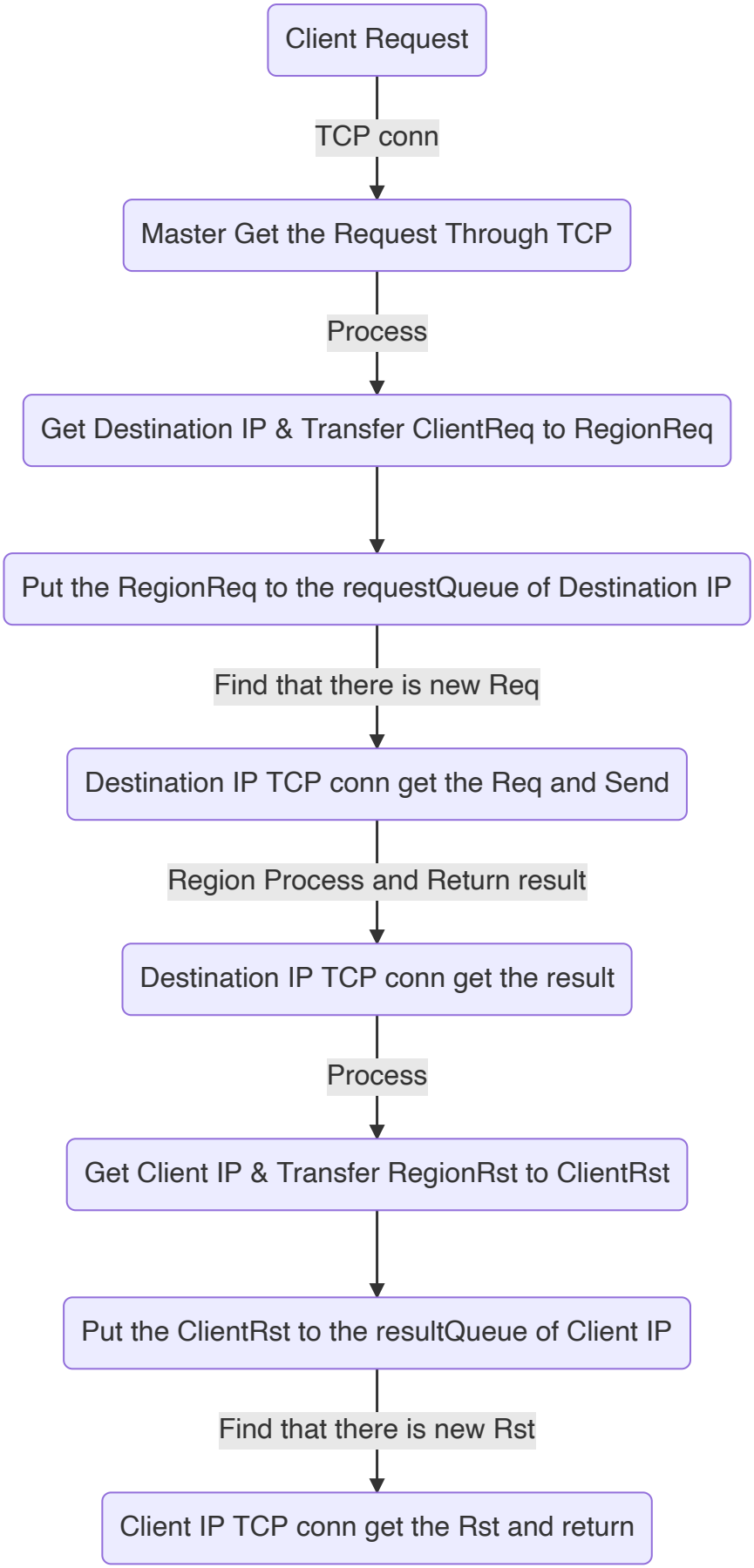
首先，需要说明的是：我们的 Client、Master Server 以及 Region Server 之间的连接是非常稳定的 Tcp 连接，其体现在 Go 语言中 就是一个 `net.Conn` 的数据类型；为了实现并发处理，我在 Master Server 中为每一个连接单独设置了一个 goroutine 来对其进行处理；而 Master Server 的一大主要功能就是对请求以及消息的转发，这就要求 Master Server 在其内部有一套复杂的 goroutine 见的通信策略以保证消息转发的通畅，在这一部分我将对该策略进行详细的介绍。

在本系统中，我选择使用 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，我定义了针对与 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：



2.2.3 Master Server 整体工作流程

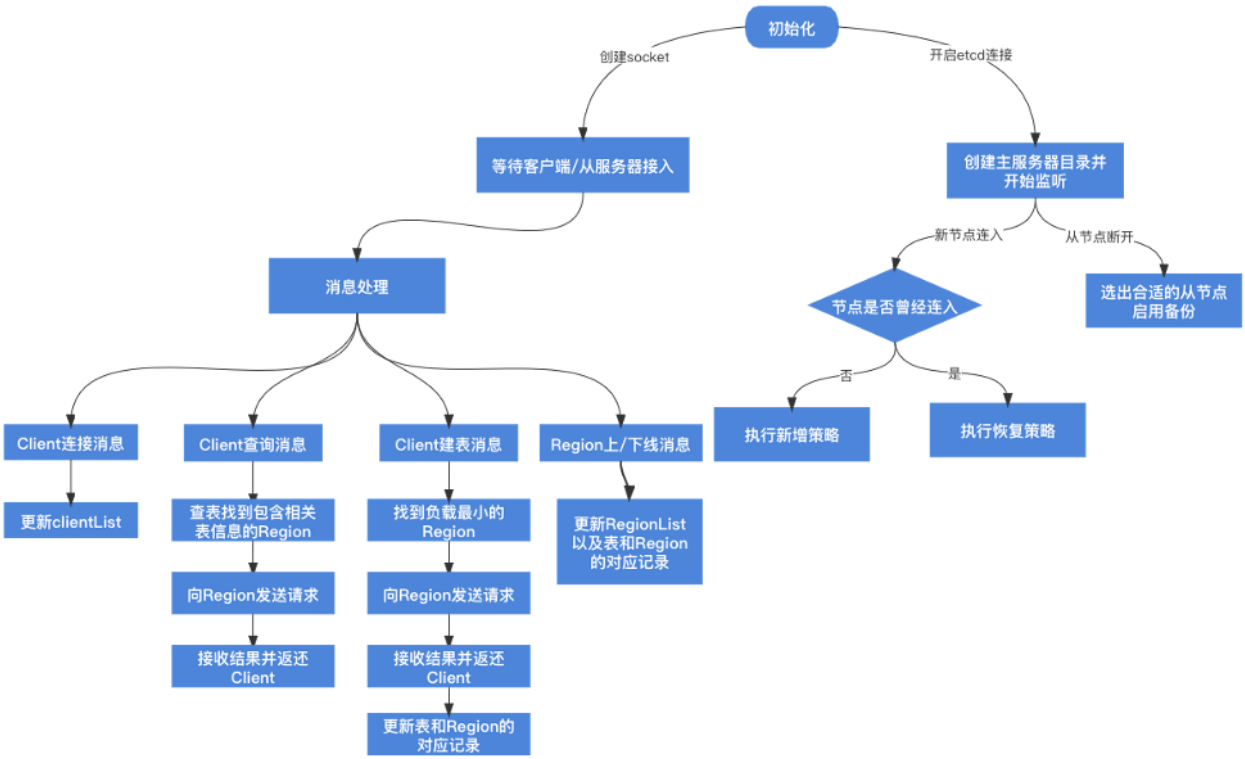


图 4 Master Server 工作流程图

3 核心模块功能设计与实现

3.1 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。

3.1.1 Create Table

前文提到，为实现负载均我在 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，其实现方式如下：

```
func (pq *PriorityQueue) getNextTwo() (string, string) {
  firstLast := (*pq)[0]
  secondLast := (*pq)[1]
  return firstLast.ipAddress, secondLast.ipAddress
}
```

3.1.2 Other Operation

由于我在 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。其部分代码实现如下：

```
// 存储  Table 与该 Table 所在 Region 对应关系
type tableInfo struct {
  tableName       string
  region_1      string
  region_2      string
}

type tableList []*tableInfo

// 获取 Table 对应的 Region
func (tq tableList) getRegionIp(tableName string) (string, string){
  region_1, region_2 := "", ""
  if index := tq.Find(tableName); index >= 0 {
    region_1, region_2 = tq[index].region_1, tq[index].region_2
  }
  return region_1, region_2
}
```

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。

3.2 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。本人负责的部分为 Copy Table 的消息转发，其代码实现如下：

```
func copyRequest(conn net.Conn) {
  for {
      select {
        case tableName := <-
regionQueue[regionQueue.find(conn.RemoteAddr().String())].copyRequestQueue  :
          fmt.Printf("> Master: Copy Table [%s].\n", tableName)
          request := regionRequest{
            TableName: tableName,
            IpAddress: "",
            Kind: "copy",
            Sql: "",
            File: nil,
          }
          fmt.Printf("> Master: Send to region(%s) [copy %s].\n",
conn.RemoteAddr().String(), request.TableName)
          msgStr, _ := json.Marshal(request)
          if _, err := conn.Write(msgStr); err != nil {
            panic(err)
```

```
        }
      }
    }
}

func forwardCopy(rec regionResult, conn net.Conn) {
  desRegion := regionQueue.getCopyRegion(conn.RemoteAddr().String())
  request := regionRequest {
    TableName: "",
    IpAddress: "",
    Kind: "new",
    Sql: "",
    File: rec.File,
  }
  regionQueue[regionQueue.find(desRegion)].requestQueue <- request
}
```

4 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 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，我充分的巩固了课上学到的分布式相关的知识，加深了自己对线程的理解，同时也对 Go 这一编程语言有了较为系统的认识，总的来说获益匪浅。