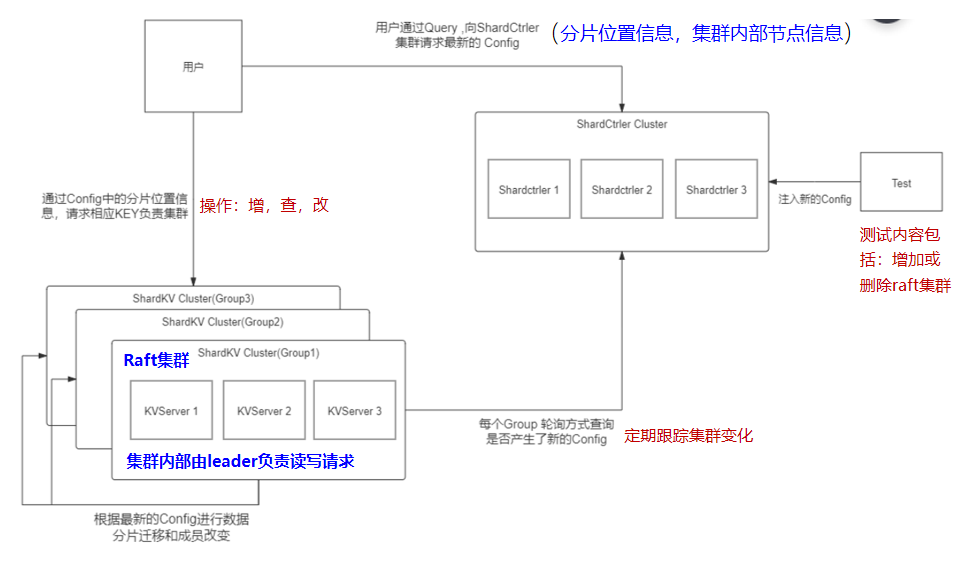
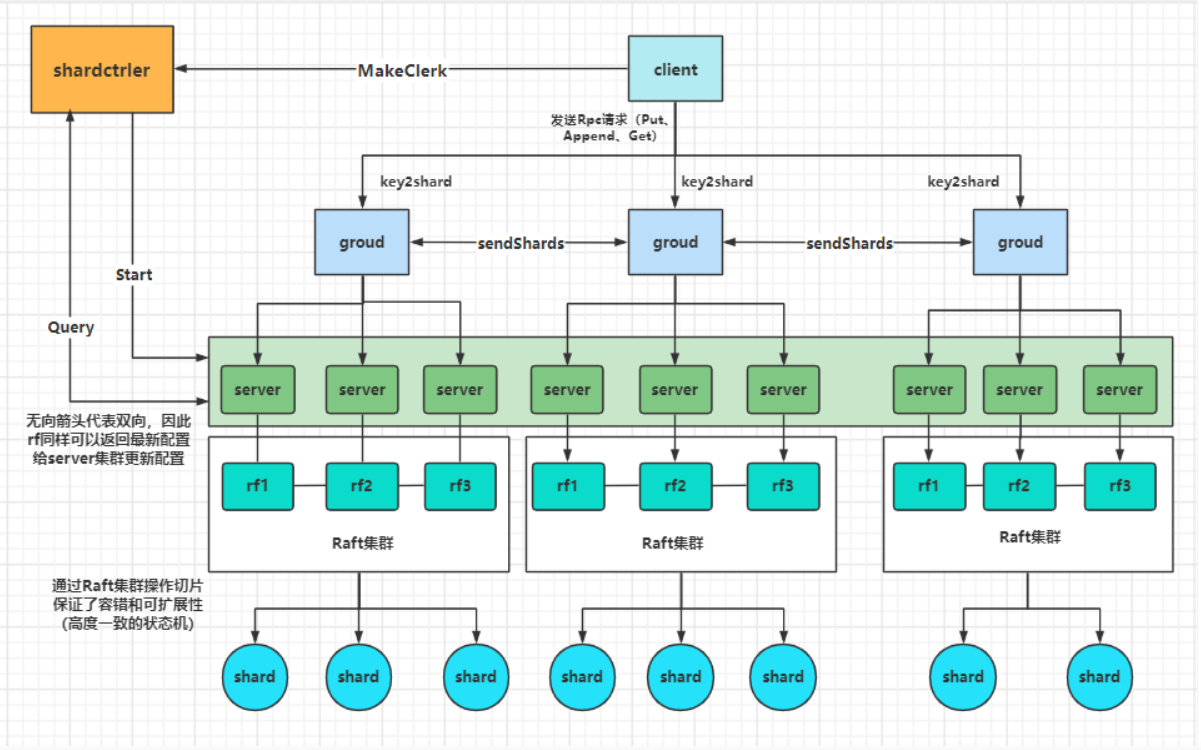
4B需要结合以下这两张图来理解（写报告时记得重新画图，不要用这些图），就是在之前4A的基础上，实现一个分片的容错数据库（跟4A那个不一样，4A的只是运行原理像数据库，4B这个才是系统中真正的数据库）。

具体来说，就是做一个key-value数据库，但是这个数据库里面的数据是分片储存的，用户可以增查改（好像不可以删）里面的数据。那么我们需要怎么找到数据存在哪个分片呢？这时候就需要我们在4A实现的那个分片控制器来管理了。

所以，整体流程是这样的，用户先向分片控制器发送一个查询请求，查到现在的分组配置信息（也就是搞清楚数据放在哪了），然后根据这个信息，去key-value数据库里面对应的位置找到想要的数据，然后再执行增查改操作。操作完了之后服务器会把新的分组配置信息发送给分片控制器。



以上是根据期末大作业PPT的理解，而咱们的代码实现思路要参考下面这个：

client客户端，将发送的请求利用Key2Shard进行分片，分到具体某个组下的server，然后这个server如果是leader则再利用自身的raft组进行共识，利用共识对整个server集群同步当前组对分片的操作，保持容错，而整个系统的集群则是通过lab4A的分片控制器来保证。

操作类型：

在分布式系统中，Put、Get 和 Append 操作通常与键值存储系统（Key-Value Store）相关。这些操作描述了对分布式键值存储中数据的写入、读取和追加行为。

Put 操作：

描述： Put 操作用于向分布式键值存储中插入或更新数据。它接受一个键值对作为输入，并将该键值对存储在系统中。如果键已经存在，则更新其关联的值。

例子： Put("key", "value") 将键为 "key" 的值设置为 "value"。

Get 操作：

描述： Get 操作用于从分布式键值存储中检索数据。给定一个键，它返回与该键相关联的值。

例子： Get("key") 返回键为 "key" 的值。

Append 操作：

描述： Append 操作用于在分布式键值存储中向现有值追加数据。它通常与字符串或列表类型的值相关联，允许在原始值的末尾追加新的内容。

例子： Append("key", "new data") 将新数据追加到键为 "key" 的值的末尾。

本文件夹代码修改的是client.go、common.go、server.go，细节如下：

client.go

Clerk结构体：这个就是客户端的结构体，主要在其中添加了seqId字段用于标识每次请求的序列号、clientId字段记录本服务器的编号，与4A中的Clerk结构图基本一样，不过这次偷懒没添加leaderId字段（加上之后可以优化WrongLeader的次数）

MakeClerk函数：用于初始化创建一个Clerk客户端实例

其他函数无可注意的修改

common.go

常量部分定义了一些错误类型和操作类型

对于各种RPC结构体，在里面添加了ClientId字段和RequestId字段用于标识客户端和请求

添加了在分组之间发送分片的RPC结构体

server.go

*结构体定义部分：*

常量设置部分设置了一些超时时间，具体去看代码中我添加的注释。

Shard结构体：表示一个分片，用于存储键值对

ShardKV结构体：定义了key-value服务器，重点在里面添加了以下字段：

·dead用来标识这个服务器是否已经停用

·Config和LastConfig用来储存当前配置和上一个版本的配置；

·shardsPersist：一个储存shard结构体的数组，用于存储分片的持久化数据

·seqMap：与4A的一样，用来进行重复性检测，防止Op被重复执行，

具体来说，seqMap 用于存储每个客户端 (ClientId) 最后一次成功执行的操作的序列号 (SeqId)。这样，服务器就可以检测到相同客户端发来的相同序列号的重复请求。

在实现中，当服务器接收到来自客户端的一个新请求时，首先会检查该请求的客户端 ID 和序列号是否在 seqMap 中存在。如果存在，说明这是一个重复的请求，可以直接忽略。如果不存在，就处理这个请求，并更新 seqMap 中该客户端的最新序列号。

·waitChMap与4A的一样，是一个等待通道，具体来说，waitChMap 的用途是：

等待通道的创建： 当服务器收到客户端请求，需要向 Raft 模块提交一个操作（Op）时，会为该操作创建一个等待通道 (chan OpReply)。

操作完成时的通知： 等待通道用于在操作完成后通知等待的线程。在执行操作的过程中，Raft 模块会将完成的操作发送到对应的等待通道，通知等待的线程操作已完成。

操作的幂等性保证： 通过 waitChMap，分片控制器可以确保每个操作只被执行一次。在操作完成后，对应的等待通道会被删除，以防止重复处理相同的操作。

在具体的实现中，当一个操作被提交到 Raft 模块时，分片控制器会创建一个等待通道并将其与操作的索引关联起来。随后，当 Raft 模块完成了对应的操作时，将结果发送到对应的等待通道，通知分片控制器操作已完成。这种机制有助于实现操作的一致性和幂等性。

·sck：一个shardctrler.Clerk客户端，用来跟分片控制器通信的

Op结构体：就是客户端发送的操作RPC包，里面定义了本次操作的类型以及一些操作序列号等相关信息。

OpReply结构体：一个简化的Op结构体，用来向客户返回结果，其中只包含了客户端标识、操作号、操作结果

*操作定义部分：*

Get函数：这个函数处理客户端的 Get 请求，首先检查所请求的键属于的分片是否在当前组内，以及该分片的数据是否已经到达。如果不满足这两个条件，则返回相应的错误。接着，构造 Get 操作的命令，通过 startCommand 函数发送该命令到 Raft 状态机，并等待命令的执行。最后，再次检查分片是否属于当前组，以及分片数据是否已经到达，如果满足条件，则将获取的值填充到 reply 中。

PutAppend函数：这个函数处理客户端的 Put 和 Append 请求，首先检查所请求的键属于的分片是否在当前组内，以及该分片的数据是否已经到达。如果不满足这两个条件，则返回相应的错误。接着，构造 Put 或 Append 操作的命令，通过 startCommand 函数发送该命令到 Raft 状态机，并等待命令的执行。最后，再次检查分片是否属于当前组，以及分片数据是否已经到达。

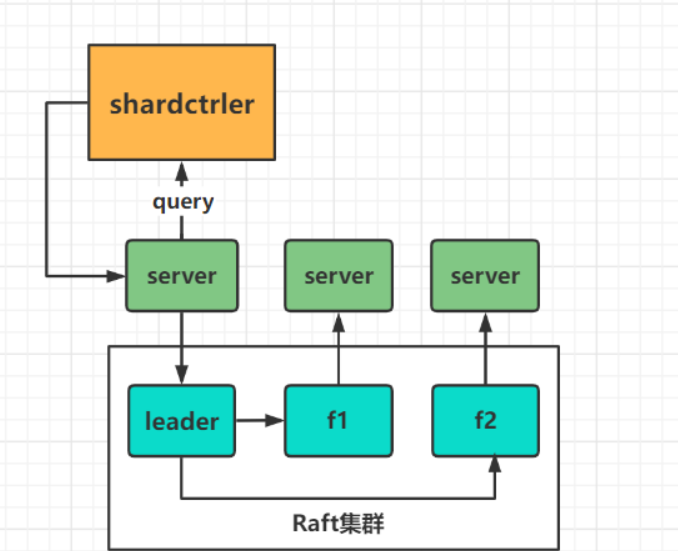
AddShard函数：这个函数用于处理来自其他服务器的 AddShard 请求。首先，构造 AddShard 操作的命令，包括客户端ID、请求ID、分片ID、分片数据和上一个应用请求的映射。然后，通过调用 startCommand 函数发送该命令到 Raft 状态机，并等待命令的执行。最后，将操作的执行结果作为错误码填充到 reply 中。

*Loop部分：*

applyMsgHandlerLoop函数：跟4A的功能基本一样，只是多加了一个处理快照的功能。这个函数用于处理 applyCh 发送过来的 ApplyMsg。在循环中，通过 select 语句监听 applyCh 和 killed 状态，根据消息类型分别处理普通的 Raft 指令和快照消息。对于 Raft 指令，根据操作类型进行相应的处理，并将结果发送到等待的通道中。对于快照消息，判断是否需要安装快照并进行数据恢复。具体来说，它处理 ApplyMsg 结构体中的命令，执行相应的操作，并将结果发送到等待通道中，以便通知发起该操作的客户端。

ConfigDetectedLoop函数：这个函数用于周期性地检测配置变化。在循环中，首先判断当前节点是否为 Leader 节点，然后检查是否已经将不属于当前组的分片分配给其他组。如果还有待发送的分片，则克隆分片数据并向目标组发送，并进行相应的 GC 操作。接着，检查是否已经接收到所有分片，如果没有，则等待下一次循环。最后，如果当前配置已经完成，轮询下一个配置，并向 分片控制器 查询下一个配置，构造 UpConfig 命令并发起。

具体流程如下图，服务器向分片控制器查询配置，然后通过raft层向所有服务器同步：



*handler部分：*

upConfigHandler函数：这个函数用于处理最新配置的更新。首先，获取当前配置和更新的配置。然后，检查当前配置号是否大于等于更新配置号，如果是则不进行处理。接着，遍历更新配置的所有分片，如果当前服务器属于更新配置的组且该分片未被分配，则初始化该分片的 KvMap，并设置 ConfigNum 为更新配置的配置号。最后，更新 LastConfig 为当前配置，更新 Config 为更新配置。

addShardHandler函数：这个函数用于处理添加分片的操作。首先，检查该分片是否已经添加或者操作是否过期，如果是则不进行处理。然后，复制操作中分片的数据到本地。最后，更新客户端序列号映射表。

removeShardHandler函数：这个函数用于处理移除分片的操作。首先，检查操作的序列号是否小于当前配置号，如果是则说明是过期的操作，不进行处理。然后，将对应分片的数据清空，并更新配置号。

*快照部分：*

PersistSnapShot函数：这个函数用于生成 kvserver 的快照数据。它创建了一个新的字节缓冲区和一个 labgob 编码器，然后使用编码器将 shardsPersist、SeqMap、maxRaftState、Config、LastConfig 编码并写入缓冲区。最后，返回生成的快照数据。

DecodeSnapShot函数：这个函数用于安装给定的快照。它首先检查快照是否为空或长度小于1，如果是，则说明是初始状态，无需处理快照。然后，创建一个新的字节缓冲区和 labgob 解码器，并使用解码器将快照数据解码到相应的变量中。最后，更新 kvserver 的状态为解码后的数据。

*工具函数部分：*

ifDuplicate函数：检查给定客户端ID和序列号是否代表一个重复的请求

getWaitCh函数：获取或创建给定索引的等待通道

allSent函数：判断是否所有切片已经发送给其他组

allReceived函数：判断是否所有切片已经收到

startCommand函数：向 Raft 发送命令并等待执行结果

cloneShard函数：根据给定的配置号和键值映射创建一个新的 Shard

*初始化部分：*

StartServer函数：创建并初始化服务器实体

Challenge部分：

这个我理解得不清楚，靠你们自行理解了

