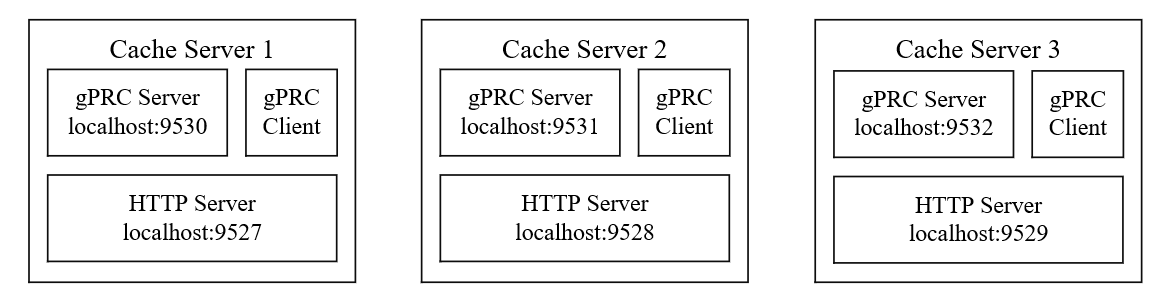
**一、总体实现**

实现的三个缓存服务器及服务器端口如下图如所示。每个服务器包含：一个HTTP服务器用于响应测试脚本中的HTTP请求、一个gRPC服务器和一个gRPC客户端用来进行服务器间通信。



每个缓存服务器的数据存储在自己的内存中。为了简化存取模型，当一个服务器通过HTTP服务器收到要保存的键值对时，会通过RPC将数据发送给另外两个服务器一同保存；当一个服务器通过HTTP服务器收到删除数据请求时，会通过RPC将删除请求一并发给另外两个服务器，因此**三个服务器中保存有同样的数据副本**。收到读数据请求时从本机内存中读取即可。

**二、HTTP接口实现**

实现HTTP接口需要注册请求处理函数并监听端口，go语言的实现方法分别为：

http.HandleFunc("/", handleHttpRequest)

http.ListenAndServe(address[0], nil)

其中address[0]为要监听的地址，handleHttpRequest即HTTP请求处理函数，结构大致为：

func handleHttpRequest(w http.ResponseWriter, r \*http.Request) {

if r.Method == http.MethodGet {

handleGet() //Get操作

} else if r.Method == http.MethodPost {

handleSet() //Set操作

} else if r.Method == http.MethodDelete {

handleDelete() //Delete操作

} else {

http.Error() //未知操作

}

}

**三、RPC实现**

每个服务器的包含一个gRPC服务端和客户端，上一节的HTTP请求会调用客户端向其他服务器的服务端发起RPC，三种HTTP请求调用客户端接口情况如下，其中RPC和用到的数据类型在cache.proto中由Protocol Buffers定义：

GET请求：因为数据在本地有完整备份，故无需与其他服务器通信；

SET请求：为保证三个服务器数据一致性，数据保存在本地内存的同时，调用另外两个服务器中gRPC服务端的SET RPC，将数据发送给另外两个服务器并保存。RPC参数为HTTP请求传入的键值对，返回值为空；

DELETE请求：为保证服务器之间的数据一致性，数据从本地内存删除的同时，调用另外两个服务器中gRPC服务端的DELETE RPC，将数据从另外两个服务器的内存中同步删除。RPC参数为HTTP请求传入Key值，返回值为删除的键值对的数量；

**四、Docker打包**

Dockerfile 构建容器的步骤：

1、复制文件到ubuntu20.04镜像，

2、安装protobuf和go编译器

3、安装go的grpc插件

4、编译proto和go源文件

compose.ymal:

设置镜像和主机共享网络，并设置镜像入口程序为编译的go程序。

**五、实验结果**

启动三个服务器：

*docker comose up*

测试脚本

*./sdcs-test.sh 3*

[运行演示视频链接](https://www.bilibili.com/video/BV1V8411r7mr/)