# 电子科技大学计算机科学与工程学院

实 验 报 告

课程名称： 分 布 式 系 统

实验名称：SDCS: Simple Distributed Cache System

学 号： 202422900232

姓 名： 王冉恒

电 子 科 技 大 学

# 实 验 报 告

## 一、 课程名称

分布式系统。

## 二、 实验项目名称

SDCS: Simple Distributed Cache System.

## 实验原理

* 该实验基于分布式系统原理，以下是该实验用到的相关原理：  
  1、分布式存储：系统有多个服务器节点可供用户存储数据，这些节点可以是物理服务器、虚拟机或容器。用户可以指定存储在哪个节点，或随机分配一个节点存储数据，同时这些数据可以被复制并在多个节点上备份，有助于提高系统的安全性，可用性。  
  2、gRPC过程调用：gRPC是一个开源的远程过程调用框架，使用HTTP/2作为传输协议，支持多种语言。它允许不同的服务通过网络相互调用，简化了分布式系统中的通信复杂性。使用Protocol Buffers（protobuf）定义服务和消息格式。通过.proto文件描述RPC服务的接口及其输入输出消息。  
  3、一致性协议：分布式系统使用统一的一致性协议来确保数据的一致性和可靠性，通过这些协议来保证存储在分布式系统中的数据都是有序可靠的，不容易出现乱码，脏数据。  
  4、缓存机制：分布式系统在内存开辟缓存空间，以此快速存储和检索数据，以提高数据访问速度。分布式缓存系统通过将缓存数据存储在多个节点上，可以提供更大的存储容量和更快的访问速度。

## 四、 实验目的

该实验旨在：

* 1. **理解分布式缓存**：通过实践，深入理解分布式缓存的概念、工作原理及其在提升系统性能和可扩展性方面的作用。
  2. **掌握缓存设计与实现**：学习如何设计和实现一个简单的分布式缓存系统，包括数据存储、数据一致性、缓存策略（如LRU、LFU等）等。
  3. **提高系统性能：**观察和分析引入缓存后，对系统性能（如访问速度、响应时间等）的改善效果。
  4. **学习分布式系统的基本架构：**了解分布式系统的组成部分、通信方式及其挑战（如网络延迟、数据一致性等）。
  5. **掌握分布式协议：**实践一些常用的分布式协议（如一致性协议、分布式锁等），提高对分布式系统复杂性的理解。

## 五、 实验内容

## 目标：完成一个简易分布式缓存系统

## 要求：

## a. Cache数据以Key-value形式存储在缓存系统节点内存中（不需要持久化）；

## b. Cache数据以既定策略（round-robin或hash均可，不做限定）分布在不同节点（不考虑副本存储）；

## c. 服务至少启动3个节点，不考虑节点动态变化；

## i. 所有节点均提供HTTP访问入口；

## ii. 客户端读写访问可从任意节点接入，每个请求只支持一个key存取；

## iii. 若数据所在目标存储服务器与接入服务器不同，则接入服务器需通过内部RPC向目标存储服务器发起相同操作请求，并将目标服务器结果返回客户端。

## d. HTTP API约定

## i. Content-type: application/json; charset=utf-8

## ii. 写入/更新缓存：POST /。使用HTTP POST方法，请求发送至根路径，请求体为JSON格式的KV内容，示例如下：

## curl -XPOST -H “Content-type: application/json” http://server1/ -d ‘{“myname”: “电子科技大学@2024”}’

## curl -XPOST -H “Content-type: application/json” http://server2/ -d ‘{“tasks”: [“task 1”, “task 2”, “task 3”]}’

## curl -XPOST -H “Content-type: application/json” http://server3/ -d ‘{“age”: 123}’

## iii. 读取缓存 GET /{key}。使用HTTP GET方法，key直接拼接在根路径之后。为简化程序，对key格式不做要求。

## 正常：返回HTTP 200，body为JSON格式的KV结果；

## 错误：返回HTTP 404，body为空。

## curl <http://server2/myname> {“myname”: “电子科技大学@2023”}

## curl <http://server1/tasks> {“tasks”: [“task 1”, “task 2”, “task 3”]}

## curl http://server1/notexistkey

## #404, not found

## iv. 删除缓存 DELETE /{key}。永远返回HTTP 200，body为删除的数量。

## curl -XDELETE http://server3/myname

## 1

## curl http://server1/myname

## #404, not found

## curl -XDELETE http://server3/myname

## 0

## 六、 实验步骤

## 操作系统：Ubuntu 20.04 开发环境：Python 3.10、Docker 24.0.7、docker-compose 1.29.2

1. **安装环境**

安装python3：

sudo apt-get install python3

安装grpcio

python3 -m pip install grpcio

安装grpcio-tools

python3 -m pip install grpcio

1. **编写proto文件并生成gRPC的代码**

Proto文件实现gRPC客户端服务端交互的消息类型和服务，内容如下：

syntax = "proto3";option cc\_generic\_services = true;

service Cache {

rpc GetCache (GetRequest) returns (GetReply) {}

rpc SetCache (SetRequest) returns (SetReply) {}

rpc DeleteCache (DeleteRequest) returns (DeleteReply) {}

}

message GetRequest {

string key = 1;

}

message GetReply {

string key = 1;

string value = 2;

}

message SetRequest {

string key = 1;

string value = 2;

}

message SetReply {

}

message DeleteRequest {

string key = 1;

}

message DeleteReply {

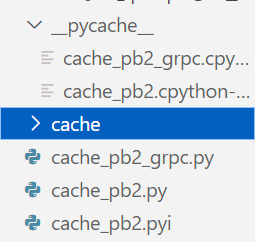
int32 num = 1;

}

使用grpc的插件生成gRPC要用到的文件：

python **-**m grpc\_tools**.**protoc **-**I**./**cache **--**python\_out**=.** **--**pyi\_out**=.** **--**grpc\_python\_out**=.** **./**cache**/**cache**.**proto

生成文件如下：



1. **实代码实现**

实现服务端gRPC服务：

class GrpcServer(pb2\_grpc.CacheServicer):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.store = {}

def SetCache(self, request, context):

self.store[request.key] = request.value

print(self.store[request.key])

return pb2.SetReply()

def GetCache(self, request,context):

print(self.store.get(request.key, 'N/A'))

return pb2.GetReply(key=request.key, value=self.store.get(request.key, 'N/A'))

def DeleteCache(self, request,context):

if request.key in self.store.keys():

self.store.pop(request.key)

return pb2.DeleteReply(num = 1)

else:

return pb2.DeleteReply(num = 0)

def StartRpcServer():

server = grpc.server(futures.ThreadPoolExecutor(max\_workers=3))

pb2\_grpc.add\_CacheServicer\_to\_server(GrpcServer(), server)

server.add\_insecure\_port(server\_Address)

print("服务端启动，ip为 ",server\_Address)

server.start()

server.wait\_for\_termination()

客户端使用http服务：

class HttpRequestHandler(http.server.BaseHTTPRequestHandler):

def do\_GET(self):

key = self.path.strip("/")

flag = 1

value = ""

for i in range(int(3)):

tmp\_address = base\_ip+str(int(i) + 8081)

channel = grpc.insecure\_channel(tmp\_address)

server = pb2\_grpc.CacheStub(channel)

value = server.GetCache(pb2.GetRequest(key=key)).value

if value != 'N/A':

flag = 0

break

if flag == 0:

self.send\_response(200)

self.send\_header('Content-type', 'application/json;charset = utf-8')

self.end\_headers()

data = {key: value}

json\_data = json.dumps(data)

self.wfile.write(json\_data.encode(encoding='gbk'))

else:

self.send\_response(404)

self.send\_header('Content-type', 'application/json;charset = utf-8')

self.end\_headers()

self.wfile.write("\n".encode(encoding='utf-8'))

def do\_POST(self):

body = self.rfile.read(int(self.headers['Content-Length']))

jsonstr = json.loads(body.decode(encoding='utf-8'))

key1 = ''

for k in jsonstr.keys():

key1 = k

value = str(jsonstr.get(key1))

channel = grpc.insecure\_channel(server\_Address)

server = pb2\_grpc.CacheStub(channel)

server.SetCache(pb2.SetRequest(key=key1, value=value))

self.send\_response(200)

self.send\_header('Content-type', 'application/json;charset = utf-8')

self.end\_headers()

self.wfile.write(json.dumps(jsonstr).encode(encoding='gbk'))

def do\_DELETE(self):

print(self.path)

key = self.path.strip("/")

self.send\_response(200)

self.send\_header('Content-type', 'application/html;charset = utf-8')

self.end\_headers()

flag = 1

for i in range(int(3)):

tmp\_address = base\_ip+str(int(i) + 8081)

channel = grpc.insecure\_channel(tmp\_address)

server = pb2\_grpc.CacheStub(channel)

value = server.GetCache(pb2.GetRequest(key=key))

if value.value != 'N/A':

server.DeleteCache(pb2.DeleteRequest(key=key))

flag = 0

if flag == 0:

self.wfile.write("1".encode('utf-8'))

else:

self.wfile.write("0".encode('utf-8'))

def StartClientServer():

handler = HttpRequestHandler

port = (int)(sys.argv[1]) + 9526

httpd = socketserver.TCPServer((base\_ip.split(':')[0],port),handler)

print("客户端启动，ip为:",base\_ip+str(port))

httpd.serve\_forever()

1. **dockers打包**

编写Dockerfile

#syntax=docker/dockerfile:1

FROM ubuntu:20.04

LABEL authors="202422900232王冉恒"# copy sourcefile to image

WORKDIR /root

COPY . .

# install python3 and protobuf compiler

RUN apt-get update

RUN apt-get install python3 -y

RUN apt-get install python3-pip -y

# install grpc plugin for python

RUN python3 -m pip install grpcio

RUN python3 -m pip install grpcio-tools

编写docker-compose.yaml

services:

node1:

build: .

command: ["python3", "test.py", "1"]

network\_mode: host

node2:

build: .

command: ["python3", "test.py", "2"]

network\_mode: host

node3:

build: .

command: ["python3", "test.py", "3"]

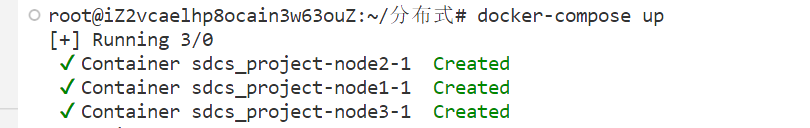
network\_mode: hostname: sdcs\_project

1. **运行程序**

在docker-compose.yaml文件所在目录下执行：

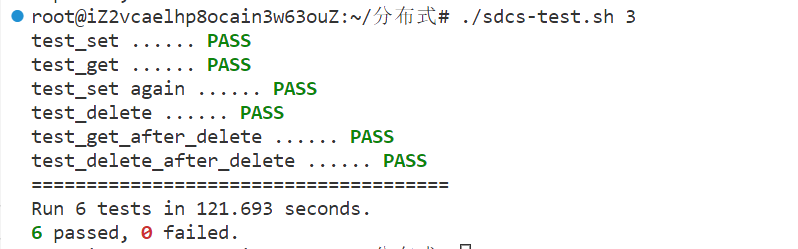
docker-compose up

运行成功结果如下：



## 七、 实验数据及结果分析

运行测试脚本，结果如下：



POST、GET、DELETE、GET AFTER DELETE、DELETE AFTER DELETE等六个操作全部成功通过，说明简易分布式缓存系统构建成功。

## 八、 实验结论

本实验包括一个分布式 KV 存储引擎从设计、开发、部署、测试的完整流程，最终顺利完成全部实验流程，设计并实现了一个功能简单的分布式 KV 存储引擎，通过测试，实现了实验目标。

## 九、 总结及心得体会

在实验过程中，我深刻体会到分布式缓存的重要性。它通过在多个节点之间存储数据，提高了读取速度，减轻了后端数据库的压力，使得系统的响应更为迅速和高效。

在实现POST、GET、DELETE等操作时，我遇到了数据一致性和并发控制的挑战。通过调试和优化代码，逐步克服了这些问题，并成功实现了不同操作间的协调和互斥，确保了缓存数据的准确性和可靠性。

总的来说，本次简易分布式缓存系统的实验让我受益匪浅。通过实际操作，我不仅掌握了基本的缓存操作，还加深了对分布式系统设计思想的理解。在未来的学习和工作中，我将继续关注分布式技术的最新发展，不断提升自己的实践和解决问题的能力。