分布式系统作业

大作业

姓名：郝裕玮

班级：计科1班

学号：18329015

目录

[1 实验项目简介 3](#_Toc93656616)

[2 环境配置 4](#_Toc93656617)

[3 实验内容 4](#_Toc93656618)

[3.1 原理分析 4](#_Toc93656619)

[3.2 代码分析 6](#_Toc93656620)

[3.3 功能测试 14](#_Toc93656621)

[4 总结 14](#_Toc93656622)

[4.1 遇到的问题和解决办法 14](#_Toc93656623)

[4.2 实验心得 15](#_Toc93656624)

# 1 实验项目简介

设计并实现一个分布式键值（key-value）存储系统，可以是基于磁盘的存储系统，也可以是基于内存的存储系统，可以是主从结构的集中式分布式系统，也可以是P2P式的非集中式分布式系统。能够完成基本的读、写、删除等功能，支持缓存、多用户和数据一致性保证。

具体要求如下：

（1）必须是分布式的键值存储系统，至少在两个节点或者两个进程中测试；

（2）可以是集中式的也可以是非集中式；

（3）能够完成基本的操作如：PUT、GET、DEL等；

（4）支持多用户同时操作；

（5）至少实现一种面向客户的一致性如单调写；

（6）需要完整的功能测试用例；

（7）涉及到节点通信时须采用RPC机制。

加分项：

（1）具备性能优化措施如cache等；

（2）具备失效容错方法如：Paxos、Raft等；

（3）具备安全防护功能；

（4）其他高级功能；

# 2 环境配置

编程语言：Python 3.7.0

第三方库：rpyc + sqlitedict

对上述2个库执行pip install xxx即可。

# 3 实验内容

## 3.1 原理分析

我实现的分布式键值系统主要由以下4个部分组成：

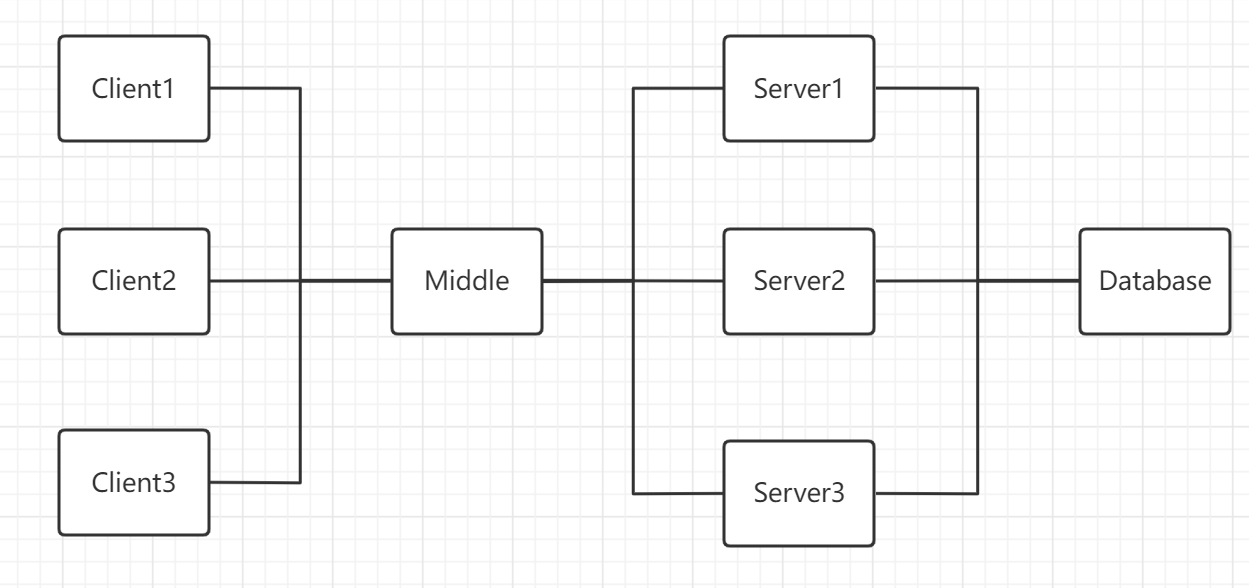
（1）客户端Client；

（2）分布式系统中间件Middle；

（3）服务器Server；

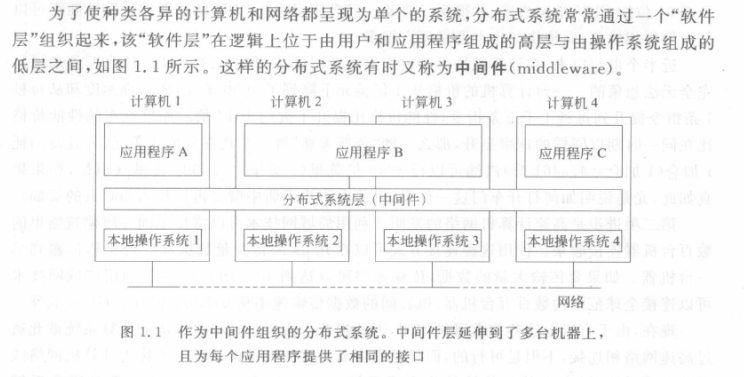
（4）数据库Database。

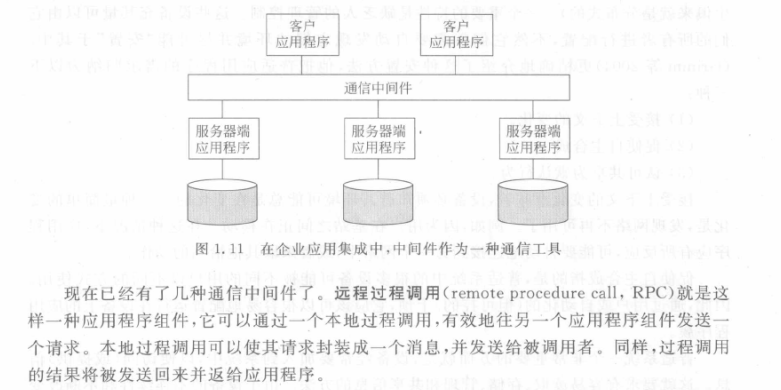
他们的关系图如下所示：



其中Client\_x通过Middle中间件与对应的Server\_x相连接：Client发出去请求给Middle中间件，Middle中间件再调用Server\_x上的功能函数，从而最终对Database进行查询或修改，并将结果返回给Client。同时由于所有Server都连接到同一个Database，所以Database的任意变化都会在该次操作结束后同步到所有Server上。

所以很容易证明我实现的分布式系统具有单调读一致性和单调写一致性。





## 3.2 代码分析

由3.1可知我们需要实现3个部分的代码：Client，Middle，Server（Database在Server中初始化并连接即可）。

具体代码如下所示（设计思路和功能分析均已包含在代码注释中）：

（1）Client.py

#coding=utf-8

import argparse

import rpyc

# argparse库:用于解析参数,并为这些参数自动生成帮助和使用信息(用于美化UI界面)

# rpyc库:它是一个Python的库,用于实现RPC和分布式计算

#Client类:客户端

class Client(object):

    #在cmd中输入help会显示如下信息

    """

    Commands Help:

    PUT key value —— Generate/Modify (key, value)

    GET key —— Query (key, value) by the key

    GETALL —— Get all (key, value) in the database

    DEL key —— Delete (key, value) by the key

    DELALL —— Delete all (key, value) in the database

    GETLOG —— Get the log

    """

    #客户端与master节点连接

    def connect(self):

        self.conn = rpyc.connect('localhost', 21000)

        #获取自己独有的客户ID

        #client要访问Master节点的代码必须通过self.conn.root.xxx才能访问

        self.id = self.conn.root.get\_id()

        return self.id

    #进入运行界面

    def run(self):

        try:

            while True:

                #输入指令

                command = input("Client %d >> " %self.id)

                #指令为help则输出指南

                if command == 'help':

                    print(self.\_\_doc\_\_)

                #反之则调用Master节点中对应的功能函数

                else:

                    msg = self.conn.root.function(self.id, command)

                    #打印结果信息

                    if msg != None:

                        print(msg)

        except KeyboardInterrupt:

            pass

#主程序

if \_\_name\_\_ == '\_\_main\_\_':

    users = {} #初始用户列表为空

    #打开文件读取

    with open('user.txt') as file:

        for i in file.readlines():

            #每一行是用户名+空格+密码,用split函数将其分开

            #users[i[0]] = i[1]代表建立字典对应值,即users[用户名]=密码

            i = i.split()

            users[i[0]] = i[1]

    #用户登录

    username = input('Please input your username:')

    password = input('Please input your password:')

    #检测用户名和密码是否匹配

    if username in users.keys() and users[username] == password:

        client = Client()

        client\_id = client.connect()

        if client\_id == None:

            print('Connection Failed.')

        else:

            print('\nWelcome to Distributed Key-Value System!\n')

            print('Your client ID is %d\n' %client\_id)

            print("Enter \"help\" for the list of commands:\n")

            client.run()

    else:

        print('Your username or password has something wrong.Please try again!')

（2）Middle.py

#coding=utf-8

import argparse

import rpyc

from rpyc.utils.server import ThreadedServer

# argparse库:用于解析参数,并为这些参数自动生成帮助和使用信息(用于美化UI界面)

# rpyc库:它是一个Python的库,用于实现RPC和分布式计算

#Middle类:分布式系统中间件

class Middle(rpyc.Service):

    #当客户端到服务器的连接建立时,on\_connect函数会被执行

    def on\_connect(self, conn):

        pass

    #当客户端到服务器的连接断开时,on\_disconnect函数会被执行

    def on\_disconnect(self, conn):

        #依次断开客户端和服务器的连接

        for temp in clients:

            temp.close()

    #为当前客户分配ID

    def exposed\_get\_id(self):

        #遍历client\_ids表,寻找尚未分配的id

        for i in range(len(client\_ids)):

            #若client\_ids[i]尚未分配则设置其为True(已使用)并返回序号i

            if client\_ids[i] == False:

                client\_ids[i] = True

                return i

        return None

    #执行各种command

    def exposed\_function(self, client\_id, clause):

        clause = clause.strip().split()

        #将command根据空格进行分割

        #strip()可消去字符串前后的空格(不包括中间)

        #split()将字符串根据空格进行分割

        lens = len(clause)

        WRONG\_MSG = 'Wrong command. Enter help if necessary.'

        if lens < 1:

            return WRONG\_MSG

        #将指令的第一个单词全部转为小写,便于后续判定

        #易知指令的第一个单词小写化后只可能是:put,get,getall,del,delall,getlog

        command = clause[0].lower()

        #开始进行条件判断

        #对于PUT key value

        if command == 'put':

            if lens == 3:

                key = clause[1]

                value = clause[2]

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                clients[client\_id].root.Put(key, value)

                #将本次操作PUT key value写到对应客户的日志中

                clients[client\_id].root.Write\_Log('PUT ('+str(key)+','+str(value)+')')

            else:

                return WRONG\_MSG

        #对于GET key

        if command == 'get':

            if lens == 2:

                key = clause[1]

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                result = clients[client\_id].root.Get(key)

                #将本次操作GET key写到对应客户的日志中

                clients[client\_id].root.Write\_Log('GET '+str(key))

            else:

                return WRONG\_MSG

            if result == None:

                return 'Key %s not found.' %key

            else:

                return result

        #对于GETALL

        if command == 'getall':

            if lens == 1:

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                #将本次操作GETALL写到对应客户的日志中

                clients[client\_id].root.Write\_Log('GETALL')

                return clients[client\_id].root.Get\_All()

            else:

                return WRONG\_MSG

        #对于DEL key

        if command == 'del':

            if lens == 2:

                key = clause[1]

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                clients[client\_id].root.Delete(key)

                #将本次操作DEL key写到对应客户的日志中

                clients[client\_id].root.Write\_Log('DEL '+str(key))

            else:

                return WRONG\_MSG

        #对于DELALL

        if command == 'delall':

            if lens == 1:

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                #将本次操作GETALL写到对应客户的日志中

                clients[client\_id].root.Write\_Log('DELALL')

                return clients[client\_id].root.Delete\_All()

            else:

                return WRONG\_MSG

        #对于GETLOG

        if command == 'getlog':

            if lens == 1:

                #client要访问服务器的代码必须通过self.conn.root.xxx才能访问

                return clients[client\_id].root.Get\_Log()

            else:

                return WRONG\_MSG

#主程序

if \_\_name\_\_ == '\_\_main\_\_':

    #(1)创建ArgumentParser()对象

    #(2)调用add\_argument()方法添加参数

    #(3)使用parse\_args()解析添加的参数

    parser = argparse.ArgumentParser()

    parser.add\_argument('--p', type=int, default=1)

    args = parser.parse\_args()

    #建立一个bool数组,初始化为args.p个False变量,表示这些id都未被使用

    client\_ids = [False] \* args.p

    #设置中间件的监听端口号用于和不同的服务器节点相连接

    clients = [rpyc.connect('localhost', 20000+i) for i in range(args.p)]

    #设置中间件与客户端连接的端口号

    middle = ThreadedServer(Middle, port=21000)

    print("Middleware is running...\n")

    middle.start()

（3）Server.py

#coding=utf-8

import argparse

import rpyc

from rpyc.utils.server import ThreadedServer

from multiprocessing import Process

from sqlitedict import SqliteDict

# argparse库:用于解析参数,并为这些参数自动生成帮助和使用信息(用于美化UI界面)

# rpyc库:它是一个Python的库,用于实现RPC和分布式计算

# multiprocessing库:用于编写多线程/多进程程序

# sqlitedict库:用于将字典信息存入数据库中,且支持多线程访问

#用于记录当前进程下的每步操作(一个客户对应一个进程)

log=[]

#Server类:服务器

class Server(rpyc.Service):

    #当客户端到服务器的连接建立时,on\_connect函数会被执行

    def on\_connect(self, conn):

        #连接建立时,创建一个字典数据库用于存储键值

        #'./database.sqlite'代表数据库文件存储路径和文件名

        #autocommit=True代表会对每次对数据库操作的结果自动提交

        self.database = SqliteDict('./database.sqlite', autocommit=True)

    #当客户端到服务器的连接断开时,on\_disconnect函数会被执行

    def on\_disconnect(self, conn):

        #pass代表不进行任何操作

        pass

    #注意,Server类中"exposed\_"开头的函数才能被客户端调用

    #调用时需将对应函数的exposed\_删去再调用

    #生成/修改键值对(key,value)

    def exposed\_Put(self, key, value):

        self.database[key] = value

    #查询键值对(key,value)

    def exposed\_Get(self, key):

        #通过key来查询对应的value

        if key not in self.database:

            return None

        else:

            return self.database[key]

    #删除键值对(key,value)

    def exposed\_Delete(self, key):

        if key not in self.database:

            #return

            return None

        else:

            #根据键值k删除键值对(key,value)

            del self.database[key]

    #获取数据库中所有键值对(key,value)

    def exposed\_Get\_All(self):

        res = [(key, self.database[key]) for key in self.database]

        res.sort()#对键值对进行字典序排序(根据key值排序)

        return res

    #删除数据库中所有键值对(key,value)

    def exposed\_Delete\_All(self):

        all\_keys = [key for key in self.database]

        for key in all\_keys:

            del self.database[key]

    #将操作记录写入日志

    def exposed\_Write\_Log(self,msg):

        log.append(msg)

    #展示日志

    def exposed\_Get\_Log(self):

        return log

#设置服务器监听端口号并运行服务器

def run(id):

    port1 = id + 20000

    server = ThreadedServer(Server, port=port1)

    try:

        #运行服务器

        server.start()

    except KeyboardInterrupt:

        server.close()

#主程序

if \_\_name\_\_ == '\_\_main\_\_':

    #(1)创建ArgumentParser()对象

    #(2)调用add\_argument()方法添加参数

    #(3)使用parse\_args()解析添加的参数

    parser = argparse.ArgumentParser()

    parser.add\_argument('--p', type=int, default=1)

    args = parser.parse\_args()

    #限制服务器在同一时间段内最多只能接受10个客户端的连接和请求

    if args.p > 10:

        raise Exception("The max number of clients is 10.")

    #创建args.p个进程

    processes = [Process(target=run,args=(i,)) for i in range(args.p)]

    print("Server is running and it can connect with %d clients at the same time." %args.p)

    #启动args.p个进程,并用join函数进行堵塞

    #join函数:在进程中可以阻塞主进程的执行,直到等待子线程全部完成之后,再继续运行主线程后面的代码

    for i in range(args.p):

        processes[i].start()

    for i in range(args.p):

        processes[i].join()

## 3.3 功能测试

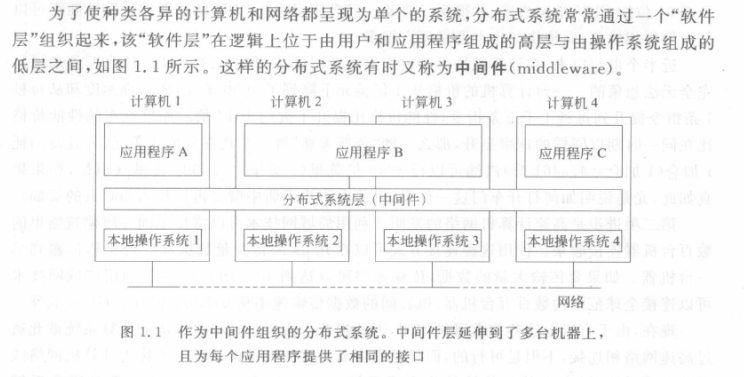
详见压缩包中“功能测试.mp4”（包含语音讲解）。

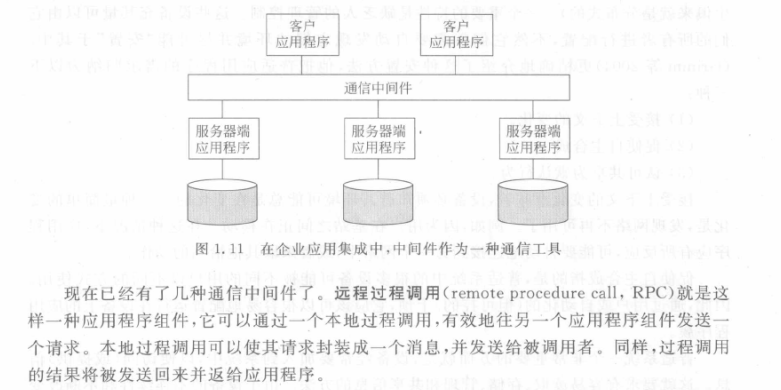
# 4 总结

## 4.1 遇到的问题和解决办法

（1）由于这次使用了python的rpyc库（Python的RPC）和sqlitedict库（Python的sqlite3 数据库的轻量级包装器，具有简单的Pythonic类dict接口并支持多线程访问），所以在调用函数和函数接口上花费了很多时间去查阅资料和学习。

（2）一开始不知道自己应该怎样设计系统结构，在重新回归书本后我得到了答案：利用中间件来连接客户端和服务器即可。





## 4.2 实验心得

经过这次实验，我对分布式系统有了更深一步的理解，明白了分布式系统在日常生活中的重要作用（比如不同客户使用不同功能时，感觉上是只和一台服务器交互，实际上是多个服务器共同运行的结果）。同时我也对RPC的原理及其运用更加熟练。

至此，本次实验圆满完成。