# **一、Memcached**

## 1.1、memcached简介

Memcached 是一个高性能的分布式[内存](http://baike.baidu.com/view/1082.htm)对象缓存系统，用于动态Web应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提供动态、数据库驱动网站的速度。Memcached基于一个存储键/值对的hashmap。其守护进程（daemon ）是用C写的，但是[客户端](http://baike.baidu.com/view/930.htm)可以用任何语言来编写，并通过memcached协议与守护进程通信。但是它并不提供冗余（例如，复制其hashmap条目）；当某个服务器S停止运行或崩溃了，所有存放在S上的键/值对都将丢失。

Memcached由Danga Interactive开发，其最新版本发布于2010年，作者为Anatoly Vorobey和Brad Fitzpatrick。用于提升LiveJournal . com访问速度的。LJ每秒动态页面访问量几千次，用户700万。Memcached将数据库负载大幅度降低，更好的分配资源，更快速访问。

## 1.2、Memcached是如何工作的

Memcached的神奇来自两阶段哈希（two-stage hash）。Memcached就像一个巨大的、存储了很多<key,value>对的哈希表。通过key，可以存储或查询任意的数据。客户端可以把数据存储在多台memcached上。当查询数据时，客户端首先参考节点列表计算出key的哈希值（阶段一哈希），进而选中一个节点；客户端将请求发送给选中的节点，然后memcached节点通过一个内部的哈希算法（阶段二哈希），查找真正的数据（item）。举个列子，假设有3个客户端1, 2, 3，3台memcached A, B, C：Client 1想把数据”tuletech”以key “foo”存储。Client 1首先参考节点列表（A, B, C），计算key “foo”的哈希值，假设memcached B被选中。接着，Client 1直接connect到memcached B，通过key “foo”把数据”tuletech”存储进去。Client 2使用与Client 1相同的客户端库（意味着阶段一的哈希算法相同），也拥有同样的memcached列表（A, B, C）。于是，经过相同的哈希计算（阶段一），Client 2计算出key “foo”在memcached B上，然后它直接请求memcached B，得到数据”tuletech”。

## 1.3、memcached提供的操作

Memcached的客户端api接口提供了如下的方法存储和获取缓存数据，并且这节方法在不同的客户端都是一致的：

get（key）:读取数据从缓存中，如果数据存在的话返回值，否则返回Null,nil,underfined。

Set(key,value [,expiry]):写入缓存数据value,如果key已经存在了就更新value值，否则就增加一个新的key/value对，如果设定了expiry值，超过设定expiry时间后键值对就失效，时间单位是秒。

Add(key,value, [,expiry]):添加键值对到缓存如果key不存在的话。

Replace(key ,value [,expiry]:替换已经存在的键值对。

Deleted（key [, time]）:从缓存中删除键，如果提供一个时间，在这段时间内这个键是处于阻塞状态。

## 二、安装部署Memcached服务

### 2.1、安装libevent库

memcached使用libevent库实现网络连接服务，理论上可以处理无限多的连接，但是它和Apache不同，它更多的时候是面向稳定的持续连接的，所以它实际的并发能力是有限制的。在保守情况下memcached的最大同时连接数为200，这和Linux线程能力有关系，这个数值是可以调整的。 关于libevent可以参考相关文档。memcachd有自己的内存分配算法和管理方式，它和共享内存没有关系，也没有共享内存的限制，通常情况下，每个memcached进程可以管理2GB的内存空间，如果需要更多的空间，可以增加进程数。

libevent下载地址：<http://down1.chinaunix.net/distfiles/libevent-2.0.16-stable.tar.gz>

# wget <http://down1.chinaunix.net/distfiles/libevent-2.0.16-stable.tar.gz>

然后是正常步骤的解压安装：

# tar zxvf lilbevent-2.0.16-stable.tar.gz

# cd libevent-2.0.16-stable

# ./configure –prefix = /usr/local/libvent-2.0.16-stable

# make

#make install

测试libevent是否安装成功：

# ls –al /usr/local/libevent-2.0.16-stable/lib | grep libevent

修改配置文件把libevent库加入ld.so.conf文件

# nano /etc/ld.so.conf 加入/usr/local/libevent-2.0.16-stable/lib

运行 ldconfig否则运行memcached会报错，提示找不到so文件

### 2.2、安装memcached:

memcached下载地址：<http://memcached.googlecode.com/files/memcached-1.4.13.tar.gz>

# wget <http://memcached.googlecode.com/files/memcached-1.4.13.tar.gz>

#tar memcached-1.4.13.tar.gz

# cd memcached-1.4.13

# ./configure

# make &make install

测试是否成功安装memcached:



### 2.3、编译安装magent代理:

magent是一款开源的memcached代理服务器软件，magent是解决memcached的单点故障的解决方案；magent的hash算法：magent采用的是:Consistent Hashing原理,Consistent Hashing如下所示：首先求出memcached服务器（节点）的哈希值， 并将其配置到0～232的圆（continuum）上。 然后用同样的方法求出存储数据的键的哈希值，并映射到圆上。 然后从数据映射到的位置开始顺时针查找，将数据保存到找到的第一个服务器上。 如果超过232仍然找不到服务器，就会保存到第一台memcached服务器上。


编译安装magent:

# wget <http://memagent.googlecode.com/files/magent-0.5.tar.gz>

# mkdir magent

# cd magent/

# tar zxvf magent-0.5.tar.gz

# cd magent/

# nano magent.c 加入如下宏定义：

(# ifndef SSIZE\_MAX

# define SSIZE\_\_MAX 32767

# endif)否则编译会报错：未指定最大值

magent.c:729: error: ‘SSIZE\_MAX’ undeclared (first use in this function)

#/sbin/ldconfig

# sed -i "s#LIBS = -levent#LIBS = -levent -lm#g" Makefile

# make

# cp magent /usr/bin/magent

测试magent代理服务器是否安装成功：

root@ubuntu:~# magent

please provide -s "ip:port" argument

memcached agent v0.5 Build-Date: Feb 21 2012 16:39:44

Usage:

 -h this message

 -u uid

 -g gid

 -p port, default is 11211. (0 to disable tcp support)

 -s ip:port, set memcached server ip and port

 -b ip:port, set backup memcached server ip and port

 -l ip, local bind ip address, default is 0.0.0.0

 -n number, set max connections, default is 4096

 -D don't go to background

 -k use ketama key allocation algorithm

 -f file, unix socket path to listen on. default is off

 -i number, set max keep alive connections for one memcached server, default is 20

 -v verbose

## 三、memcached集群测试

### 3.1、测试环境

1.Memcached提供了很多客户端api，可供java,c,c++,python…的客户端的调用，在这里我们用python这门语法优雅功能强大的语言进行此次测试。

首先安装python lib库，运行如下命令：

# sudo apt-get install python

自动安装完成后，默认版本为2.7.2。

测试是否正确安装：



接着安装python-memcached-latest.tar.gz.这是memcached的客户端api，安装好了之后直接用python操作调用memcached。

下载地址：ftp://ftp.tummy.com/pub/python-memcached/python-memcached-latest.tar.gz

安装如下：

# tar –zxvf python-memcached-latest.tar.gz

# cd python-memcached-latest

# python setup.py install

测试安装是否成功：



若 import memcache没有报错则导入无误。

2.memcached环境已经部署完毕，接下来分别在四台虚拟linux服务器上运行测试任务：

分别在四台台主机上启动配置相同的memcached服务：





在主机192.168.1.108上启动magent代理服务器:

其中-s代表主服务器，-b 192.168.1.111：11211为备份服务器。

我为此写了一个python客户端程序(Memcached.py)调用magent代理往三个主服务器写入和读取数据。

### 3.2、测试脚本源码

'''

Created on 2012-2-22

@author: Administrator

'''

'''

multiprocess :

current instance

# Memcached –d –u root –m100 –l 192.168.1.108 –p 11211

# Memcached –d –u root –m100 –l 192.168.1.109 –p 11211

# Memcached –d –u root –m100 –l 192.168.1.110 –p 11211

# Memcached –d –u root –m100 –l 192.168.1.111 –p 11211

magent proxy

# magent –u root –n 4096 –l 192.168.1.108 –p 11212 –s 192.168.1.108:11211

–s 192.168.1.109:11211 –s 192.168.1.101:11211 –b 192.168.1.111:11211

'''

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

 import memcache

 key = None

 value = None

m = []

# 用代理节点实现数据写入和读取

 magent = memcache.Client(['192.168.1.108:11212'],debug = 1)

 for i in range(8): #测试向服务器写入8个数据

 key = 'key'+str(i)

 value = 'value'+str(i)

 magent.set(key,value)

 m.append(key)

 server\_one = memcache.Client(['192.168.1.108:11211'],debug = 1

 print("server\_one store list:")

 for keys in m:

 values =server\_one.get(keys)

 print(values)

server\_two = memcache.Client(['192.168.1.109:11211'],debug = 1)

 print("server\_two store list:")

 for keys in m:

 values = server\_two.get(keys)

 print(values)

 server\_three = memcache.Client(['192.168.1.110:11211'],debug = 1)

 print("server\_three store list:")

 for keys in m:

 values = server\_three.get(keys)

 print(values)

 back\_up\_server = memcache.Client(['192.168.1.111:11211'],debug = 1)

 print("back\_up\_server store list:")

 if back\_up\_server is not None：

 print "backup is not none"

 for keys in m:

 values = back\_up\_server.get(keys)

 print(values)

运行程序：

# python Memcache.py

server\_one store list:

None

value1

None

None

value4

None

None

value7

server\_two store list:

value0

None

value2

None

None

value5

None

None

server\_three store list:

None

None

None

value3

None

None

value6

None

back\_up\_server store list:

value0

value1

value2

value3

value4

value5

value6

value7

### 3.3、测试结论

测试结果和预期的一样，magent通过consistant hase算法根据key值的不同分别把value写入主服务器，如果备份服务器只有一个的话，把所以key\_value映射到同一个备份服务器，相同的，我们可以启动几个备份服务器，这时候magent根据consistent hase 算法和写入主服务器一样的方式分别写入备份服务器。

当用命令杀死（kill）一个memcache主服务进程之后，如果通过magent代理节点取数据不会出现问题，结果和上面的一样。但是当我们把kill掉的memcache复活，再尝试取结果则出现故障，因为magent不会再从备份节点取数据，而是直接去复活的那个节点拿数据，而恢复的节点内无数据故出现读取None值的问题，这个问题有待解决。

# 二、Redis

## 2.1、Redis简介

Redis: **RE**mote **DI**ctionary **S**erver(Redis) 是一个由Salvatore Sanfilippo写的key-value存储系统。Redis提供了一些丰富的数据结构，包括lists, sets, ordered sets 以及hashes 还有和Memcached一样的strings结构.Redis当然还包括了对这些数据结构的丰富操作。redis是一个高性能的key-value数据库。redis的出现，很大程度补偿了memcached这类key-value存储的不足，在部分场合可以对关系数据库起到很好的补充作用。redis提供了Python，Ruby，Erlang，PHP等客户端，使用很方便Redis的优点：

性能极高 Redis能支持超过 100K+每秒的读写频率。

丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。

原子性Redis的所有操作都是原子性的，同时Redis还支持对几个操作全并后的原子性执行。

丰富的特性 Redis还支持 publish/subscribe, 通知, key 过期等等特性.

## 2.2、Redis常用的操作

Redis提供了string,set,list数据类型及操作：

strings数据类型及操作：

和memcached一样提供set,get操作。

Set key ‘value’

Get key

Set集合数据类型及操作：

SADD key member — 增加元素

　　SREM key member — 删除元素

　　SCARD key — 返回集合大小

　　SISMEMBER key member — 判断某个值是否在集合中

　　SINTER key1 key2 ... keyN — 获取多个集合的交集元素

SMEMBERS key — 列出集合的所有元素

Redis的set是string类型的无序集合。set元素最大可以包含(2的32次方)个元素。

list数据类型及操作：

LPUSH key string — 将某个值加入到一个key列表头部

LPUSH key string — 将某个值加入到一个key列表末尾

LLEN key — 列表长度

LRANGE key start end — 返回列表中某个范围的值，相当于mysql里面的[分页](http://baike.baidu.com/view/159980.htm)查询那样 LTRIM key start end — 只保留列表中某个范围的值

LINDEX key index — 获取列表中特定索引号的值，要注意是O(n)复杂度

LSET key index value — 设置列表中某个位置的值

　　LPOP key 抛出表头值

　　RPOP key — 和上面的LPOP一样，就是类似栈或队列的那种取头取尾指令，可以当成消息队列来使用了.

## 2.3、安装配置Redis

进入Redis管网: http://redis.io/download

# wegt <http://redis.googlecode.com/files/redis-2.4.4.tar.gz>

# tar zxvf redis-2.4.2.tar.gz
# cd redis-2.4.2
# make
# make install
# cp redis.conf /etc/

第二步：修改配置

#vi /etc/redis.conf

我是按照下面这个博客上面的指导配置的：

http://blog.csdn.net/21aspnet/article/details/6960757

#### 2.3.1、下载安装redis-python客户端

# sudo apt-get install python-redis 其实就是一个python文件（模块）。

检测是否安装成功成功：

Import redis没有报错就说明成功安装。

启动并测试redis服务器：



启动成功，现在使用redis存储key-value.



存取成功.

#### 2.3.2、配置redis分布式、主从同步

Redis和memcached一样实现了服务器的集群，在redis上是主从同步。

配置slave服务器如下：

我是在同一台主机上修改配置文件，当然多机环境也一样。

首先，复制redis.conf文件，命名为redis-slave.conf:



编辑：

# nano /etc/redis-slave.conf修改如下：



绑定从服务器端口：



绑定ip地址：127.0.0.1

设置从服务器的备份DB：



添加主服务器ip:port确认从服务器监听哪台主服务。



配置完成保存退出。

启动主从、服务器成功：



在python脚本中测试是否主从一致：



得到想要的值，配置redis的主从服务器成功。这里只是实现了简单的主从同步，redis的功能很全也很强大，分布式集群的实现需要深入学习后再实施。

# 三、Memcahced、Redis做mysql数据库缓存

## 3.1、安装mysql数据库

Ubuntu中安装mysql的方式很多，如在如源码编辑安装、deb包安装等。这次是我进行进行二进制版本安装的。

安装步骤：

# http://downloads.mysql.com/archives.php -- 二进制mysql下载链接

# mysql-5.1.54-ubuntu.tar.gz --下载安装文件。

# tar –zxvf mysql-5.1.54-ubuntu.tar.gz –C /usr/local

# cd /usr/local

# mv myssql-5.1.54-ubuntu.tar.gz mysql -将此目录命名为mysql

# cd mysql

# groupadd mysql

# useradd –g mysql mysql

# chown -R mysql .[注意小数点]—让该用户mysql对该目录用户使用权

chgrp -R mysql .

bin目录很重要，里面包含了mysql客户端和服务器，把它添加到环境变量中

nano /etc/profile –进入编辑状态

添加如下行

export PATH=$PATH:/usr/local/mysql/bin

完成后按住Ctrl+O进行保存，然后再按住Ctrl+x 退出

在scripts/下有个安装脚本，我们用它安装mysql

scripts/mysql\_install\_db --user=mysql --basedir=/usr/local/mysql --datadir=/usr/local/mysql/data

--第一个mysql:指是所属用户

--第二个mysql:指安装路径

--basedir:指安装路径

--datadir：指数据存放路径

让root拥有此目录的权限，当然除data外

chown -R root . --目录所属root用户

chown -R mysql data –数据所属mysql用户

bin/mysqld\_safe --user=mysql & --启动mysql

3.设置开机自动启动。

每次敲mysqld\_safe –user=mysql &很烦。接下来让它开机自动运行。

在support-files里有个mysql.server脚本，这就是启动脚本，只要把它放到系统启动自动运行目录里，就可以让mysql自动运行。

切换到系统启动脚本目录

cd /etc/init.d –自动启动目录

拷贝mysql.server并重命名为mysql

cp /usr/local/mysql/support-files/mysql.server mysql –拷贝文件并重命名

chmod +x /etc/init.d/mysql --让其具有运行权限

cd /etc/rc2.d

建立init.d/mysql的符号链接

ln -s /etc/init.d/mysql /etc/rc2.d/S20mysql

好了，全部完成，重启（reboot）。

## 3.2、安装MySQL-python库

要使用python连接mysql数据库需要一个MySQLdb,这个就好比java的jdbc，是一个用python写的api 库，下面是安装配置过程：

下载地址：<http://ncu.dl.sourceforge.net/project/mysql-python/mysql-python/1.2.3/MySQL-python-1.2.3.tar.gz>

# tar –zxvf MySQL-python-1.2.3.tar.gz

# cd MySQL-python-1.2.3

#在编译安装之前先把Python开发包安装好这个很重要！不然gcc报错，如下：

# sudo apt-get install python-dev

安装完成后继续执行如下命令：

MySQL-python-1.2.3# python setup.py build

MySQL-python-1.2.3# python setup.py install

安装完测试如下：



导入MySQLdb无误，说明安装成功了。

## 3.3、测试过程

#### 3.3.1、redis vs memcached

首先测试在单线程下memcahced和redis缓存读写速度并分析数据得到结论，再测试在多线程下模拟并发时memcached/redis缓存读取数据的情况，所有的测试数据都由随机数函数控制产生，运行环境一样：

Configration:

Cpu:双核Intel® Atom™ CPU D525 @1.80GHz,内存2G,100M网卡OS:ubuntu11.10-server-x64

测试键/值长度为32 bytes

分别启动一个memcached、redis服务进程：

Memcached –d –m268 –uzdxhave –l192.168.1.109 –p11211

Redis-server /etc/redis.conf





Redis配置图：



测试结果，源代码如下

在单线程情况下：



 图3.3.1



 图3.3.2

由上图3.3.2和3.3.2可以看出,在单线程条件下，相同的写入和读取数据次数 memcached的写入和读取时间都要优于redis缓存而且两者性能都比较稳定，没有出现连接服务器失败的情况。

接下来是在多线程情况下的测试，流程如下：在固定写入数量（2000）的条件下，开启多个线程同时连接（每个线程2000次写入/读取）memcached、redis服务器，统计结果如下：



 图3.3.3

我在这次实验用的是多线程下模拟并发的环境，从图3.3.3实验数据和图表看来memcached表现很稳定，写入和读取时间相差不是很大，但是redis在低并发的时候表现比的比memcached好，当并发数超过10，就出现连接不稳定，连接不上服务器的情况，而memcached很适合那种高性能多并发的环境下。

结论：memcached是一个高性能多并发的缓存系统，适合简单的追求高性能，有独立服务器，和大内存的环境下，然而redis性能也不错，它提供更多的对内存数据的操作方式，可以自定义内存数据格式，适合有特殊需求，可备份内存数据，而它自身的支持分布式的更多功能还在开发之中。

#### 3.3.2搭载memcached/redis缓存的mysql性能测试

测试流程：首先，在mysql数据库上新建一个数据库，数据库名test,在里面创建一个mytable表，表字段为num varcahr(50)，val varchar(50):



接下来在单线程下分别对其读、写速度进行测试，写入数据结果图如下：



 图3.3.4

从图3.2.4可以看出在没有搭载缓存服务器的情况下，mysql\_set(草绿色)的写入性能是最好的，而且稳定行很高，因为当有新数据写入时是没有经过两次写操作（一次写入缓存），而是直接写入后台数据库，所以性能是最高的。

当把memcached/redis当做mysql的缓存服务器后，写入数据用时基本是单独不用缓存的两倍多一点，而且，redis(红色)做为缓存写入性能比memcached的要好。

接下来是各服务器读取数据的统计图表，如下图3.3.5：



 图3.3.5

 如上图3.3.5所示，在读取数据少于2000的时候，mysql\_get,mem\_mysql\_get,redis\_mysql\_get的性能都差不多，随着读取数据量的增大，mysql\_get性能直线下降，到了用户不能忍受的地步，而搭载了缓存的mysql,在读取数据少于10000的情况下读取性能非常稳定，而且速度比较理想。Memcached配合mysql和redis配合mysql在少于15000的读取数据量的情况下，性能差不多，但随着数据量的增大，redis-mysql相比memcached-mysql的性能下降的比较严重。

## 3.4、实验结论

在此次实验环境，在单线程条件下，单独测试memcached&redis服务器的性能，相同的写入和读取数据次数memcached的写入和读取性能都要优于redis缓存。在多线程模拟多并发的测试中memcached性能也较redis稳定。

然而，当memecached做为数据库缓存，读取数据速度是三者中最快的，而redis做为数据库缓存写入数据的性能略比memcached的高，搭载缓存服务器后测试mysql的写入速度下降严重，因为数据经过了两次写入操作，但是读取速度明显加快。在写入数据时客户端编码性能有待提高，在客户端实现的时候，这些应该是可以优化的。

## 3.5、测试源代码

**#!/bin/env python/mysql-memcached**

from threading import Thread

import sys

import memcache

class Workers(Thread):

 def \_\_init\_\_(self, c, times, gs):

 Thread.\_\_init\_\_(self)

 self.c = c

 self.times = times

 self.gs = gs

 self.completiontime = 0

 self.name = c

 def run(self):

 import MySQLdb

 from hashlib import md5

 from time import time

 startTime = time()

 for i in range(self.times):

 conn = MySQLdb.connect(host='localhost', user='root', passwd='star', db='test', port=3306)

 mem = memcache.Client(['192.168.1.109:11211'],debug=1)

 cursor = conn.cursor()

 hashval = md5(str(self.c) + str(i)).hexdigest()

 if self.gs == 'set':

 if mem.get(hashval):

 pass

 else:

 cursor.execute('insert into mytable values("'+hashval+'","'+hashval+'")')

 mem.set(hashval,hashval,60)

 else:

 row = mem.get(hashval)

 if row:

 # print "get from memcached"

 # row = mem.get(hashval)

 pass

 else:

 # print "get from mysql"

 cursor.execute('select \* from mytable where num = "' + hashval + '"')

 row = cursor.fetchone()

 conn.commit()

 cursor.close()

 conn.close()

 endTime = time()

 self.completiontime = endTime - startTime

#chars = '1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20'.split(',')

threads = []

for char in range(int(sys.argv[3])):

 print char

 gs = 'get'

 if sys.argv[1] == 'set':

 gs = 'set'

 timee = int(sys.argv[2])

 t = Workers(char, timee, gs)

 threads.append(t)

 t.start()

for t in threads:

 t.join()

 print 'Got ', t.name, t.gs, t.completiontime

**#!/bin/env python/mysql**

from threading import Thread

import sys

class Workers(Thread):

 def \_\_init\_\_(self, c, times, gs):

 Thread.\_\_init\_\_(self)

 self.c = c

 self.times = times

 self.gs = gs

 self.completiontime = 0

 self.name = c

 def run(self):

 import MySQLdb

 from hashlib import md5

 from time import time

 startTime = time()

 for i in range(self.times):

 conn = MySQLdb.connect(host='localhost', user='root', passwd='star', db='test', port=3306)

 cursor = conn.cursor()

 hashval = md5(str(self.c) + str(i)).hexdigest()

 if self.gs == 'set':

 cursor.execute('insert into mytable values("'+hashval+'","'+hashval+'")')

 else:

 cursor.execute('select \* from mytable where num = "' + hashval + '"')

 row = cursor.fetchone()

 conn.commit()

 cursor.close()

 conn.close()

 endTime = time()

 self.completiontime = endTime - startTime

#chars = '1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20'.split(',')

threads = []

for char in range(int(sys.argv[3])):

 print char

 gs = 'get'

 if sys.argv[1] == 'set':

 gs = 'set'

 times = int(sys.argv[2])

 t = Workers(char, times, gs)

 threads.append(t)

 t.start()

for t in threads:

 t.join()

 print 'Got ', t.name, t.gs, t.completiontime

**#!/bin/env python/mysql-redis**

from threading import Thread

import sys

import redis

class Workers(Thread):

 def \_\_init\_\_(self, c, times, gs):

 Thread.\_\_init\_\_(self)

 self.c = c

 self.times = times

 self.gs = gs

 self.completiontime = 0

 self.name = c

 def run(self):

 import MySQLdb

 from hashlib import md5

 from time import time

 startTime = time()

 for i in range(self.times):

 conn = MySQLdb.connect(host='localhost', user='root', passwd='star', db='test', port=3306)

 red = redis.Redis('localhost')

 cursor = conn.cursor()

 hashval = md5(str(self.c) + str(i)).hexdigest()

 if self.gs == 'set':

 if red.get(hashval):

 # print "key-value in redis\_memery"

 pass

 else:

 cursor.execute('insert into mytable values("'+hashval+'","'+hashval+'")')

 red.set(hashval,hashval)

 else:

 row = red.get(hashval)

 if row :

 pass

 # print "get from redis"

 # row = red.get(hashval)

 else:

 print "get from mysql"

 cursor.execute('select \* from mytable where num = "' + hashval + '"')

 row = cursor.fetchone()

 conn.commit()

 cursor.close()

 conn.close()

 endTime = time()

 self.completiontime = endTime - startTime

#chars = '1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20'.split(',')

threads = []

for char in range(int(sys.argv[3])):

 print char

 gs = 'get'

 if sys.argv[1] == 'set':

 gs = 'set'

 times =int(sys.argv[2])

 t = Workers(char, times, gs)

 threads.append(t)

 t.start()

for t in threads:

 t.join()

 print 'Got ', t.name, t.gs, t.completiontime