Modern Database Concepts
Prof. Dr. Florian Heinz
florian.heinz@oth-regensburg.de
Exercise Sheet 8

OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Exercise 1: Install and start a local Redis cluster instance

On the CIP pool hosts, boot Ubuntu, start a shell and change into the directory **/data/temp** (your home directory is on a network share and does not provide enough space)

Download the Redis source tar.gz-archive from the Redis homepage and unpack it to **/data/temp**

Compile it with `make`

Use the "create-cluster" script in the "utils/create-cluster" directory to **start** redis nodes and **create** a cluster from these

Now use the redis-cli client to connect to the local instance at port 30001.

Exercise 2: First tests

Set the key **hello** to the value **world** and fetch it again

Set an expiry time for the key **hello**. Find out, what the flags NX XX GT LT are for and test one of them. There is a command to display the expire time, which comes in handy here.

Simulate a fair worker queue with a Redis *list*. A fair queue does not honor priorities or the duration of the individual work items. Insert three work items work1-work3 into a list called **workers** with a single command, then display the whole list. After that, append a fourth item to the tail. Get the item at the front of the list without removing it. Then, pop it from the front.

Try out one of the blocking commands, for example **BLPOP**. Open a second terminal and also connect to the node 30001. Perform the BLPOP command on the list until it blocks using a timeout of 5400. In the other terminal, push an item into the list then and keep an eye on both terminals to see what happens.

Now, similarly, implement a priority queue with a **sorted set**. Priority queues append their worker items with an associated priority value. By convention, the items with lower priority values have higher priority in most implementations.
Again, use one terminal as the producer and the other one as the consumer. Insert three worker items into the priority queue with three different priority values. On the consumer terminal, display the sorted set with the item of highest priority at the top. Then use a suitable (blocking) command to retrieve the items with highest priority one by one until the call blocks. Then, push another item into the list in the producer terminal

Find out the coordinates of the **OTH (campus)**, **Dom**, **Bismarckplatz** and **Koewe (center)** and save these into a geospatial key **pois** (for points of interest). What's the distance of the Dom to the other members? Which members are inside a 500m radius around Bismarckplatz? Display the raw values of the items with sorted-set functions

Exercise 3: Cluster

Try to add a key **peter** as a hash value with a field age: 42. Observe the message from the server and retry on the correct shard.

You now want to set the key **susan** to the value **miller**. Do not just attempt to do that to see the "MOVED" message, but try to find out, which shard is the correct one, by looking at the cluster map (help @cluster) and determining the hash slot for "susan" with the help of https://crccalc.com/

Now, find out which node is the secondary of the node on which you have just set the value "susan". Connect to it and read the value from there.

Try to set it on the secondary, what happens?

In another terminal, find out the PID of the primary node (use the command line tools "ps ax" and "kill -9 <pid>" for that) and kill it the hard way. Wait a few seconds and retry to set the value for "susan".

Check the current cluster map. Now restart the missing node (just do a **create-cluster start** again). Connect to that node and check the cluster map again. What is the status of the node? How can you promote it to the **master** role again? (see help @cluster). Check the cluster state again.

There is a mode for redis-cli to automatically follow redirects if a key is not local. Find it and try it out.

Exercise 4: LUA Scripting

Write a simple LUA script that just returns the result of 3*7

Write another script that takes a key argument. You can assume, that a name (string) is stored in it. The script should change the value of the key to 'Hello Name'. You can use **redis.call('CMD', arg1, arg2...)** to execute redis commands inside a lua script.

Now, write a script that takes a key name as key argument, e.g. **task1**. The script should then look into the keys **{task1}op1** and **{task1}op2**. In these keys, numbers should be stored. The script now should do the operations +, -, *, / on the two operands and store the results in **{task1}add**, **{task1}sub**, **{task1}mul** and **{task1}div**

. So, the key **task1**, that is passed to the script, is indeed empty (and unused), only the derived keys are used.

Exercise 5: External programming

In this task we will try to connect to a redis cluster with a python client

To install the required modules on the CIP-Pool hosts:

```
wget https://bootstrap.pypa.io/get-pip.py
    python3 get-pip.py
    python3 -m pip install redis
```

Start a python interpreter and import the redis module

Connect to the cluster at bag.sysv.de:6379

Increment the counter "visitors" by one and push your (nick-)name into the list "iwashere"

Redis also supports cached LUA scripts. You can do a "SCRIPT LOAD <script>" and get back a SHA1-Hashvalue. This script is then cached and can be called with "EVALSHA <hash> ..." as in this example:

```
> EVAL "return 'Hello ' .. ARGV[1]" 0 "World"
"Hello World"
> SCRIPT LOAD "return 'Hello ' .. ARGV[1]"
"3f7f08e8f1ce304ab5855ab647e539fb1663c01b"
> evalsha 3f7f08e8f1ce304ab5855ab647e539fb1663c01b 0 "World"
"Hello World"
```

On the server "bag.sysv.de" a script with SHA1-Hash "b0d1733bfc87a2de9671a2bd97fdbab856fc7b32" is cached. Try to find out, what it does.