分布式系统实验报告

2023 年 11 月 25 日

1 实验目的

完成一个简易分布式缓存系统。

2 实验内容

实现一个简易分布式缓存系统，该系统应满足以下条件：

1. 缓存数据以 Key-Value 形式存储在缓存系统节点的内存中（不需要持久化）。
2. 缓存数据根据一定策略（如 round-robin 或 hash）分布在不同节点（不考虑副本存储）。
3. 服务至少启动 3 个节点，不考虑节点动态变化。
   (a) 所有节点均提供 HTTP 访问入口
   (b) 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；
   (c) 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。
4. HTTP API 约定
   (a) Content-type: application/json; charset=utf-8
   (b) 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
   (c) 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
   (d) 删除缓存 DELETE /key。永远返回 HTTP 200，body 为删除的数量。
5. 程序基于 docker 打包，并通过 docker compose 启动运行（每个 cache server 为一个 docker 实例）。compose.yaml 能直接启动不少于规定数量的 cache server。每个 server 将内部 HTTP 服务端口映射至 Host，外部端口从 9527 递增

3 实验原理

本实验旨在通过构建一个高性能的分布式缓存系统，深入探讨和展示高效缓存管理以及在分布式环境下对数据的快速检索技术。该系统基于 Go 语言构建，融合了多项关键技术和组件，以优化数据的存储和访问过程。

3.1 关键技术和组件

1. 缓存机制: 采用最近最少使用（LRU）算法为基础管理缓存数据，确保高效利用缓存空间，优先移除长时间未被访问的项。在此基础上增加了删除功能，能够删除缓存的指定节点。
2. 一致性哈希算法: 使用一致性哈希来均匀分配缓存键到各个节点，实现负载平衡和有效的数据检索。
3. Singleflight 机制: 通过 Singleflight 组件减少重复的数据请求，提高了系统的整体效率和响应速度。
4. HTTP 通信模块: 实现节点间的数据交换和通信，支持分布式结构中的数据同步。
5. 不可变数据视图: ByteView 组件提供了对缓存数据的安全和不可变的访问，保障数据完整性。
6. 缓存基本操作: 实现了缓存的添加、查询和自动淘汰功能，支持高效的数据处理。

4 实验步骤

4.1 写入/更新

命令行输入示例：

```
curl -XPOST -H "Content-type: application/json" http://server1/ -d '{"myname": "电子科技大学@2023"}'
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}'
```

分别测试在三个不同端口进行缓存的写入

4.2 读取

```
curl http://server2/myname
curl http://server1/tasks
curl http://server1/notexistkey
```

分别测试在不同端口进行读取，以及测试查找不存在的缓存

4.3 删除缓存

```
curl -XDELETE http://server3/myname
curl http://server1/myname
curl -XDELETE http://server3/myname
```

测试在其他端口能否删除缓存，以及删除后再次查找的情况

4.4 使用测试文件 sdcs-test.sh 对系统进行测试

该测试文件在 test_set ，test_get ，test_set again ，test_delete ，test_get_after_delete, test_delete_after_delete ，六个部分进行测试，综合分析系统对数据的处理功能。

5 实验数据及结果分析

5.1 写入/更新

通过在终端输入指令，返回的值均为” save current port “，说明写入成功

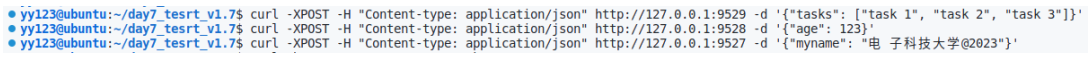

图 1: POST 输入指令

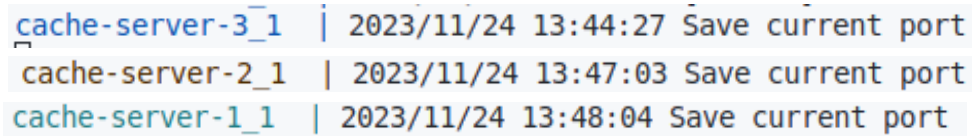

图 2: POST 返回值

5.2 读取

如下图所示，输入 GET 命令后，均可以正常返回输入的键值对。

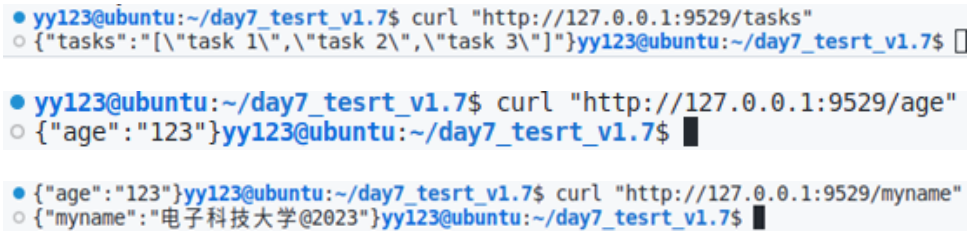

图 3: GET 命令执行情况

5.3 删除缓存

如下图所示，首先删除本地缓存再进行查找；接下来在其他端口进行两次删除后进行查找。第一次的 delete 返回结果为 1，说明删除成功；查找结果为 Not Found，说明已成功删除

第二次结果 delete 返回为 1，说明删除成功，第二次 delete 返回 0，说明内存中已没有该缓存。最后通过 GET 命令进行查找，查找结果为 Not Found，该缓存已被成功删除。

```
yy123@ubuntu:~/day7_tesrt_v1.7$ curl -XDELETE http://127.0.0.1:9529/tasks
1yy123@ubuntu:~/day7_tesrt_v1.7$ curl "http://127.0.0.1:9527/tasks"
Not Found

yy123@ubuntu:~/day7_tesrt_v1.7$ curl -XDELETE http://127.0.0.1:9529/age
1yy123@ubuntu:~/day7_tesrt_v1.7$ curl -XDELETE http://127.0.0.1:9527/age
0yy123@ubuntu:~/day7_tesrt_v1.7$ curl "http://127.0.0.1:9527/age"
Not Found
```

图 4: DELETE 命令执行情况

5.4 测试脚本执行结果

将脚本中 MAX_ITER 变量设为 500 进行测试，结果如下图所示：上图结果表明可以成功通

```
yy123@ubuntu:~/day7_tesrt_v1.7$ sudo ./sdcs-test.sh 3
test_set ..... PASS
test_get ..... PASS
test_set again ..... PASS
test_delete ..... PASS
test_get_after_delete ..... PASS
test_delete_after_delete ..... PASS
======================================
Run 6 tests in 122.097 seconds.
6 passed, 0 failed.
```

图 5: 脚本执行结果

过该测试脚本，该系统的主要功能通过测试。

6 实验结论

通过上述实验步骤测试，该分布式系统能够成功实现本地的数据缓存，获取，删除。另外在节点间也可以成功通过 http 进行数据缓存，获取和删除。