DDBS课程项目报告

顾掀宇 gxy21@mails.tsinghua.edu.cn
仲逊 zhongx21@mails.tsinghua.edu.cn

摘要

本项目目标是构建一个大数据管理系统并在分布式环境下部署，以解决真实世界问题，我们以MongoDB和HDFS作为底层数据库，用Golang语言在此之上构建系统，支持了数据批量写入，更新以及查询。我们还提供了一个CLI（命令行控制界面），用户可以通过输入命令以进行查询、监控等任务。我们使用Docker Compose并创建多个容器以模拟真实的分布式环境，实验证明我们的系统可以完成上述目标。

关键词: 分布式数据库，Golang, MongoDB, HDFS, Docker

引言

在当今，数据源源不断地从各种地方产生：社交媒体，在线购物...，数据处理规模也从过去的TB量级逐渐扩展到了PB量级，我们愈发需要高效的数据管理系统以管理这些数据。上个世纪，一台机器上部署的数据库系统即可满足绝大部分需求，而如今将数据管理系统部署到分布式环境早已是主流。而不同的场景数据规模和种类有所差别，对其数据管理系统也提出了不同的要求。例如一些分析型任务对于时延并无过多要求，而对于抢票、购物系统则对系统并发性和数据一致性提出了更高要求。

本文所述的系统指结合已有的多种类型的数据库/文件存储系统（如MySQL，PostgreSQL, HDFS等），封装并统一它们的访问接口以为用户提供更高层次服务的系统，通常这类系统根据不同的需求而进行定制。

问题定义

本文的场景以及所要处理的数据则是来自于新闻网站，数据集同时包含了结构化与非结构化数据，结构化数据包括包含用户信息和文章信息，此外还记录了一系列用户阅读文章的记录，包括阅读时长、分享评论等信息。非结构化数据则是指文章对应的文章内容、图片等信息。所要实现的系统目标列举如下：

1. 分布式环境下的数据加载（需要考虑到数据切分与数据复制）
2. 数据插入、更新与查询的高效执行
3. 监控数据库服务器的运行情况以及其所管理的数据等
4. 系统的错误恢复与错误处理
5. 系统的可扩展性（包含数据库的扩展以及迁移等）

相关工作

首先是关于结构化数据的存储。关系型模型是一类经典的数据存储模型，于上个世纪提出。关系型数据库在市场长期占有相当的份额，经典代表产品如MySQL与PostgreSQL。而NoSQL(Not Only SQL)是一项全新的数据库革命性运动，NoSQL则发展出了多种类型的数据库产品，包括kv键值存储、文档存储以及图形数据库等。其中，适合本文应用场景的是文档存储，不同于传统关系型数据库中的表以及每行对应的元组，文档存储对应的概念为集合-文档，集合对应表，文档可以视作一个json对象存储在集合之中，这一类数据库以MongoDB作为代表。

对于非结构化存储如图片、视频等，一个普遍的选择是使用HDFS，HDFS是Hadoop的分布式文件系统，以实现大规模数据可靠的分布式读写。它可以做到做到对于分布在多个机器上的存储的同一访问。

为了决定DBMS产品/技术选型，我们需要再一次明确任务需求：

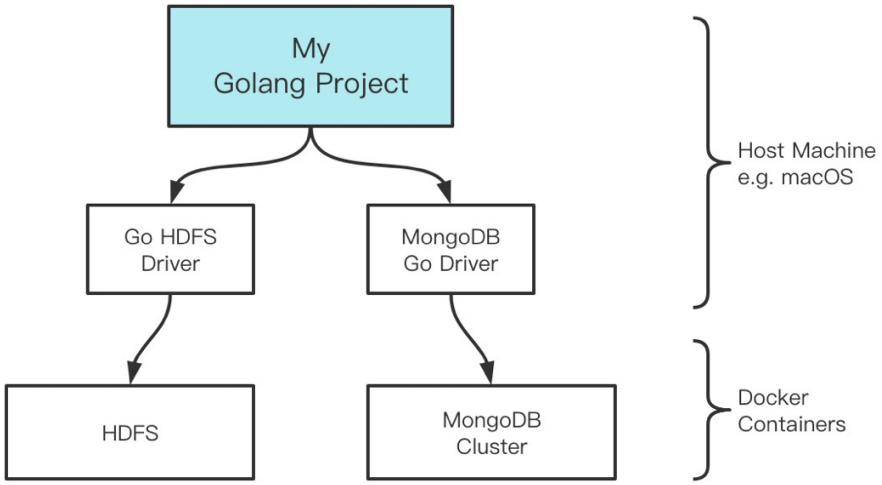
1. 不需要ACID事务支持
2. 分析型任务居多，数据一旦被载入后不需要频繁的后续修改
3. 对于时延没有提出高要求
4. 关注数据的分布式存储（数据分片以及复制）

其中关系型数据库的代表产品如MySQL，PostgreSQL，近年来的新兴产品TiDB等也获得一些良好口碑。其中MySQL，PostgreSQL是单机数据库，需要额外的中间件来实现分布式，如Vitness，部署起来较为麻烦，而TiDB则原生支持关系型数据库的分布式部署，但是其目前来说还未成为主流数据库，故暂不在选择列表里。

另一方面文档数据库的代表产品MongoDB，则很好的支持了分布式部署，其官方文档里详细描述了分布式部署以及相关的配置，同时我们更多关注数据的存储，因此MongoDB会是一个很好的选择。

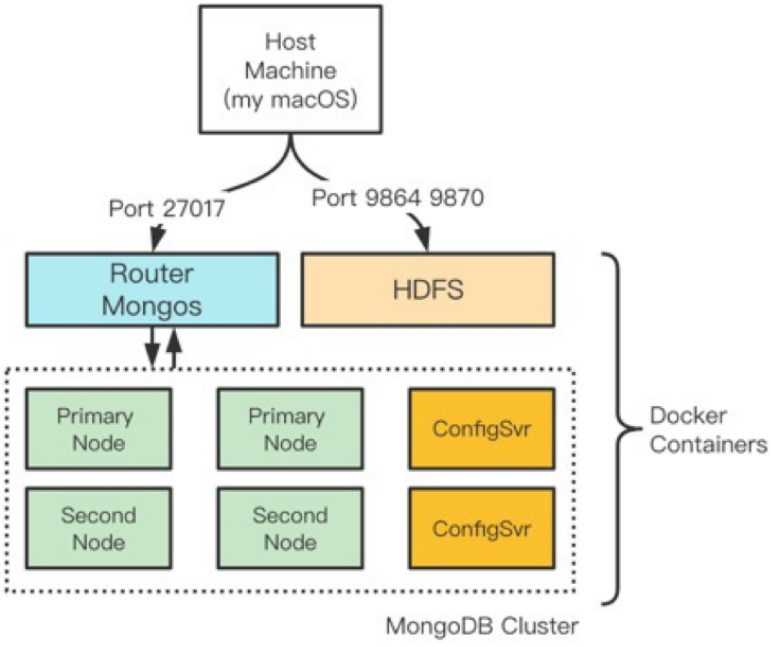
我们最终使用MongoDB和HDFS分别用于我们存储结构化与非结构化数据。

系统概览



可以用以上的图概括本文的系统结构，系统利用官方提供的Go HDFS驱动以及MongoDB Go驱动和底层部署的HDFS和MongoDB集群交互，其中HDFS与MongoDB集群部署在Docker容器中，而我们的项目前端可以运行在主机上（例如macOS），通过Docker容器暴露的端口进行通信。

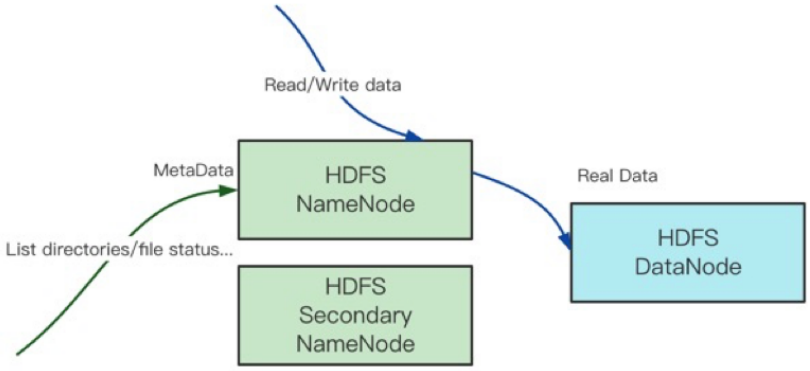
建立分布式环境



我们可以使用Docker Compose来完成分布式环境的部署，步骤划分如下：

1. 创建一个Docker网络空间192.168.1.0/24，然后创建多个容器给它们分配不同的静态IP，这样可以模拟出当我们购买云服务器集群时处理网络问题的情景
2. Docker容器的分配情况如上图所示，由于本机资源所限，我们可以使用一个节点模拟HDFS集群，并使用7个节点创建一个MongoDB集群并给它们分配不同角色。
3. 我们只需要暴露MongoDB的router节点的端口27017，以及HDFS的两个端口，将另外的网络通信细节隐藏在Docker网络之下，关于MongoDB和HDFS的不同角色将于后文介绍。

HDFS集群

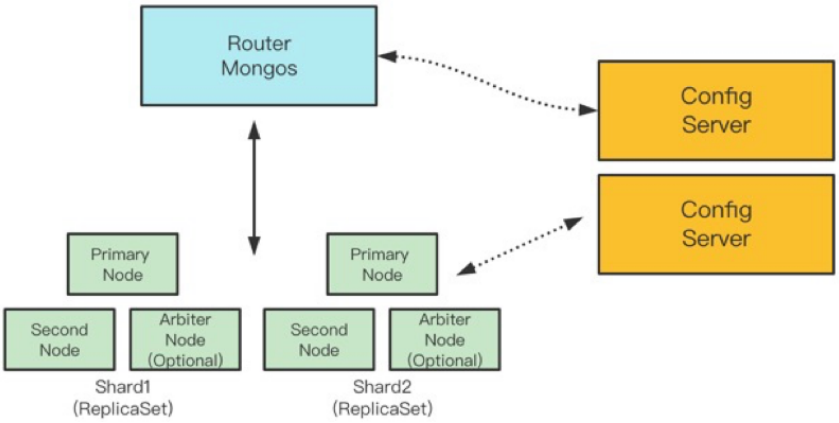


HDFS集群中有三种不同角色的节点，其中不同节点作用如下：

- NameNode维护了整个文件系统的文件目录树，文件目录的原信息以及文件的数据块索引。同时它还协调客户端对于文件的访问。
- DataNode存放着文件系统实际的数据
- SecondaryNode相当于NameNode的辅助，它用来定期把NameNode上的一些文件（如日志）合并到自己身上，避免NameNode的日志过大影响性能。

当有客户端请求来临时，如上图所示，如请求命令是查询目录/文件信息等，则名字节点可以直接返回结果，而如果请求是对于文件进行读写等，则NameNode会把请求转发至对应的DataNode节点进行操作。而在实际使用过程中，HDFS相当于对于多台机器的存储资源进行了整合，那么实际上我们就可以把它看作一个黑盒，它提供了明确的输入和输出（这个黑盒的交互模式）以供用户来使用。对于Golang HDFS Driver使用，除了预先需要连接到HDFS集群，实际使用思路和操作本机的文件也并无二致。

MongoDB集群

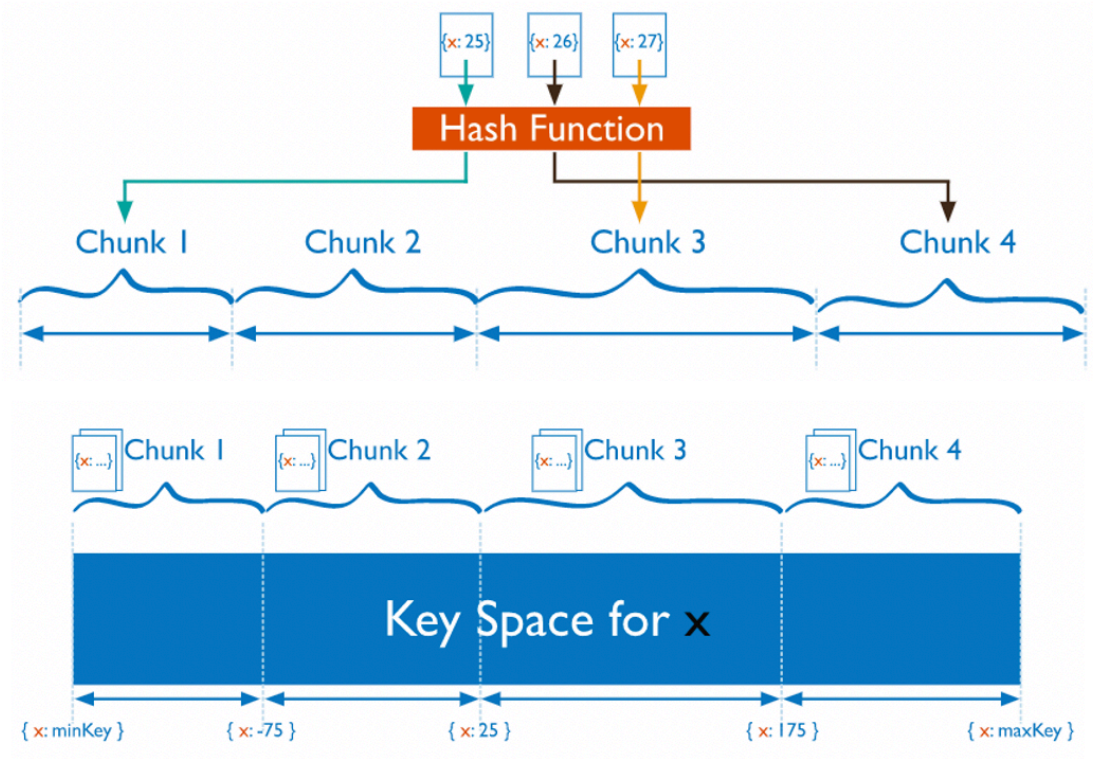


MongoDB集群中包含三种角色如下:

1. Mongos(Router)，为客户端提供接口，可以将客户端的请求转发给对应的集群机器上
2. Shard，分片，用于存储数据，对应通过部署多个分片可以完成数据库的水平扩展，其中每个分片可以部署为一个复制集（Replica Set），一个复制集中可以包含主节点、从节点以及仲裁节点，同一个复制集内部维护了相同的一部分数据，通过数据冗余以及高可用性可以达到错误恢复的能力。
3. Config Servers，配置服务器，存储了系统的一些元数据以及一些设置。

在测试的过程中，我们可以使用一个路由节点，一个或两个配置服务器，以及两个分片（每个分片包括一对主从节点）来完成整个集群的配置。

MongoDB分片设置

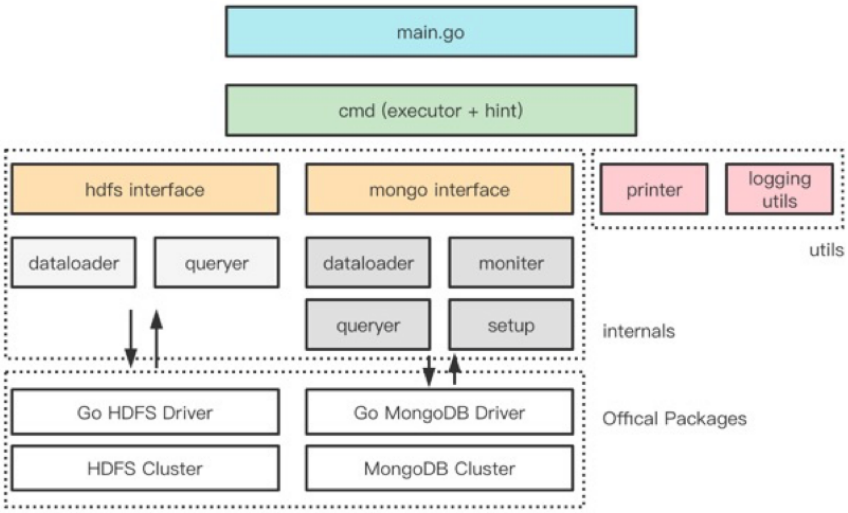


MongoDB分片方式主要分为两种：

1. Hashed Sharding，将输入数据通过一个哈希函数映射到不同范围并切分成数据块
2. Ranged Sharding，将输入数据通过原始的数据范围进行划分并切分成数据块

此外我们还可以根据分片键创建分片苏剧区域，将每个区域与群集中的一个或多个分片相关关联起来。稍微将展示在本文问题设定下该如何配置分片。

项目结构



使用Go语言实现部分的项目结构可以用上图来表示，从上而下来看，各部分功能列举如下：

1. main.go: 整个系统的入口
2. cmd：存放了CLI的实现（包含了执行器与补全器）
3. internal
    - hdfs：与hdfs通信的代码，包含了读取文件列表，读/写文件等
    - mongo：与MongoDB集群通信的代码
4. utils：包含了日志模块与打印模块，是系统的辅助功能

如上图所示，mongo模块下面包含几部分具体的实现，包含数据加载、查询、监控等。

接口示例

hdfs模块和mongo模块提供的接口分别如下所示，可以看到打印状态、查询以及数据加载接口等。

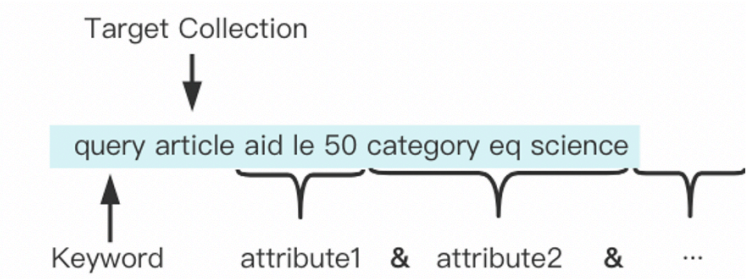
```
// mongo
func PrintCollectionStats(colls []string)
func PrintShards()
func QueryData(collectionName string, andConditions []Cond, showDetails bool)
func LoadData(target string)
func ShardingSetup()
// hdfs
func GetPathInfo(path string) ([]gowfs.FileStatus, error)
func GetArticleImages(aid string) []image.Image
func GetArticleContent(aid string) string
```

其中QueryData函数作为所有接口的核心，函数内部的调用示例如下，共接收两个参数，第一个参数是集合名称，第二个参数接收一个过滤器列表，可以看到图中示例列出了三个条件对于返回结果进行过滤。而在命令行进行使用时，由于MongoDB本身并未提供类似于SQL的结构化查询语句，于是在命令行我们使用如下的语句进行查询。第一个query表面这是一个查询一句，第二个字段表示待查询的集合，后面每三个为一组，表示针对某一个属性进行过滤。

function call inside of the code

```
_, _ = manager.QueryData( collectionName: "article", []mongo.Cond{
    { Field: "aid",  Op: mongo.OpCompGE,  Val: "1000"},
    { Field: "aid",  Op: mongo.OpCompLE,  Val: "1005"},
    { Field: "title",  Op: mongo.OpCompEQ,  Val: "title1002"},
})
```

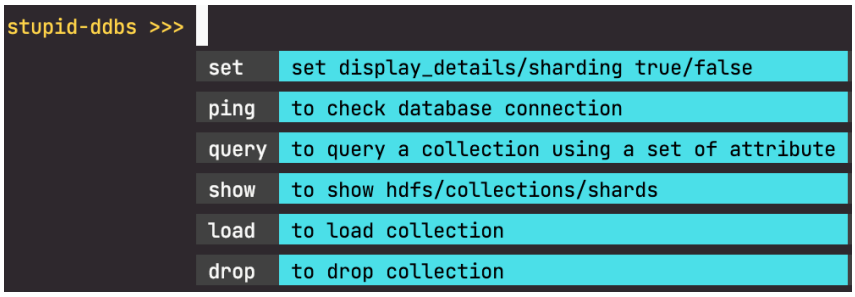
Using CLI Command



运行结果与性能

运行演示

部分运行的结果展示如下，此外，由于命令行界面无法展示图片，故我们使用了image2acsii，将图片转化为彩色acsii码字符串展示图片，可以通过`set display_details true;`命令来设置是否显示文章的内容和图片。



命令行提示与自动补全界面

```
stupid-ddbs >>> query user uid le 20 region eq Beijing;
```

TIMESTAMP	ID	UID	NAME	GENDER	EMAIL	PHONE	DEPT	GRADE	LANGUAGE	REGION	ROLE	PREFERTAGS	OBTAINEDCREDITS
2000-Jan-01	u0	0	user0	male	email0	phone0	dept15	grade3	en	Beijing	role0	tags18	97
2000-Jan-01	u3	3	user3	male	email3	phone3	dept9	grade2	en	Beijing	role1	tags30	55
2000-Jan-01	u4	4	user4	female	email4	phone4	dept18	grade1	zh	Beijing	role2	tags39	94
2000-Jan-02	u6	6	user6	female	email6	phone6	dept10	grade2	zh	Beijing	role2	tags31	90
2000-Jan-03	u11	11	user11	male	email11	phone11	dept15	grade4	zh	Beijing	role2	tags11	95
2000-Jan-03	u12	12	user12	female	email12	phone12	dept12	grade4	zh	Beijing	role1	tags39	99
2000-Jan-03	u13	13	user13	male	email13	phone13	dept6	grade3	zh	Beijing	role0	tags3	38
2000-Jan-03	u15	15	user15	male	email15	phone15	dept10	grade3	zh	Beijing	role2	tags41	73
2000-Jan-03	u16	16	user16	male	email16	phone16	dept9	grade4	zh	Beijing	role1	tags48	44
2000-Jan-04	u18	18	user18	male	email18	phone18	dept17	grade3	zh	Beijing	role2	tags33	49
2000-Jan-04	u20	20	user20	male	email20	phone20	dept3	grade4	zh	Beijing	role0	tags11	82

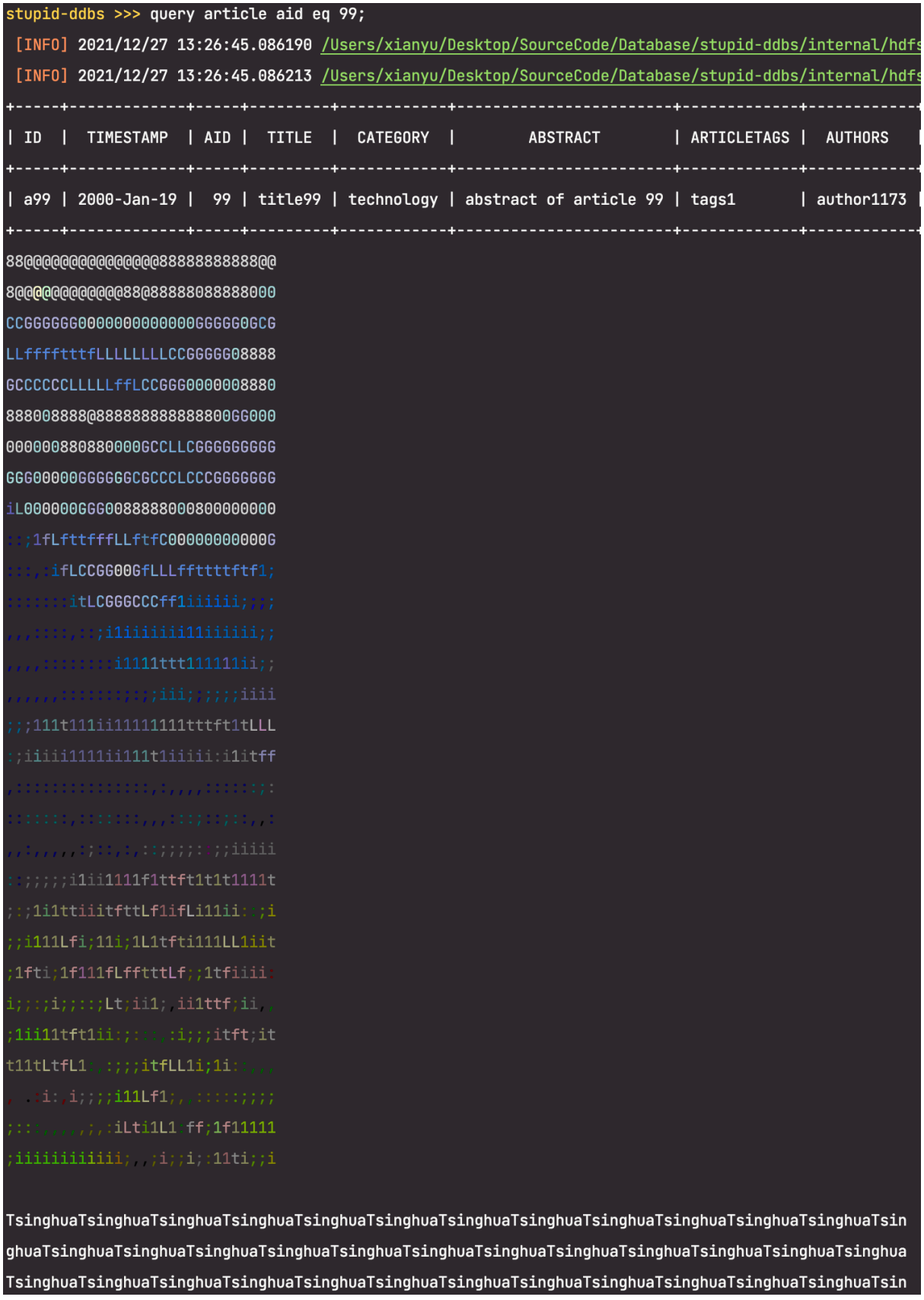
```
exec time  15.226028ms
```

使用两个过滤条件对于user表进行查询

```
stupid-ddbs >>> query popular granularity eq year;
```

TIME	GRANULARITY	ARTICLE AIDS
2004-Jan-01	year	8876 8992 8687 8066 9149
2000-Jan-01	year	1958 452 523 777 880
2001-Jan-01	year	2552 2708 2422 3099 3718
2002-Jan-01	year	5109 4941 4907 4442 4817
2003-Jan-01	year	6258 7513 6115 7275 7123

对于popular表进行查询



展示文章内容和使用image2ascii码显示图片

数据分片

数据分片结果如图所示，其中user按照region进行分片，article按照category进行分片配置，可以看到命令行打印出来的结果显示配置分片的集合（user，beread，read和article）对比未分片（popular）的显示效果。此外还可以在mongosh中用MongoDB自带的命令`sh.status()`命令查看分片状态。可以看到user已经成功按照区域进行配置。

```
stupid-ddbs >>> show collections;
 [INFO] 2021/12/27 13:36:20.835053 /Users/xianyu/Desktop/SourceCode/Database/stupid-
 [INFO] 2021/12/27 13:36:20.851191 /Users/xianyu/Desktop/SourceCode/Database/stupid-
  NAMESPACE          SHARDED  SHARDS  SIZE       COUNT    AVGOBJSIZE  STORAGESIZE
  ProjectDB.user     true     Total   2887589    10000    287.8018    2256896
                              rs1     1725176    5991     287         1871872
                              rs2     1162413    4009     289         385024
  ProjectDB.beread   true     Total   23143970   10000    2314.0      15486976
                              rs1     23143970   10000    2314        15486976
  ProjectDB.read     true     Total   227004333  1000000  226.481702  169459712
                              rs1     109565088  481702   227         87150592
                              rs2     117439245  518298   226         82309120
  ProjectDB.article  true     Total   3090401    10000    308.6245    1126400
                              rs1     1681093    5415     310         614400
                              rs2     1409308    4585     307         512000
  ProjectDB.popular  false    rs1     47933      327      146.0       36864
exec time  96.866506ms
```

命令行界面显示

```
'ProjectDB.user': {
  shardKey: { region: 1 },
  unique: false,
  balancing: true,
  chunkMetadata: [ { shard: 'rs1', nChunks: 4 }, { shard: 'rs2', nChunks: 1 } ],
  chunks: [
    { min: { region: MinKey() }, max: { region: 'Beijing' }, 'on shard': 'rs1', 'last modified': Timestamp({ t: 2, i: 1 }) },
    { min: { region: 'Beijing' }, max: { region: 'Beijinh' }, 'on shard': 'rs1', 'last modified': Timestamp({ t: 1, i: 2 }) },
    { min: { region: 'Beijinh' }, max: { region: 'Hong Kong' }, 'on shard': 'rs1', 'last modified': Timestamp({ t: 1, i: 3 }) },
    { min: { region: 'Hong Kong' }, max: { region: 'Hong Konh' }, 'on shard': 'rs2', 'last modified': Timestamp({ t: 2, i: 0 }) },
    { min: { region: 'Hong Konh' }, max: { region: MaxKey() }, 'on shard': 'rs1', 'last modified': Timestamp({ t: 1, i: 5 }) }
  ],
  tags: [
    {
      tag: 'BJ',
      min: { region: 'Beijing' },
      max: { region: 'Beijinh' }
    },
    {
      tag: 'HK',
      min: { region: 'Hong Kong' },
      max: { region: 'Hong Konh' }
    }
```

user表的分片和数据块情况

执行时间

任务（加载）	执行时间
载入user表（10000条）	635.678ms
载入article表（10000条）	835.862ms
载入read表（100000条）	28.539s
计算beread表	1h10m22.000s
计算popular表	8.977s

任务（查询）	执行时间
查询user表1条（query user uid eq 1;）	11.135ms
查询user表10条（query user uid le 10;）	13.498ms
查询user表100条（query user uid le 100;）	26.374ms
查询user表100条（query user uid le 100;）	139.242ms

任务（HDFS）	执行时间
载入100条文章数据到HDFS	1m13.542s
载入500条文章数据到HDFS	5m19.431s
读取一篇文章	9.950ms
读取一篇文章及其内容和图片（使用HDFS）	258.674ms

讨论

基础功能

1. 分布式环境的部署（MongoDB与HDFS）
2. 使用CLI可以进行MongoDB数据的加载与查询，HDFS数据的写入与查询
3. 完成数据分片的配置
4. 可以通过CLI完成MongoDB分片状态的读取，HDFS路径查询等
5. 若想要进行数据的更新以及更多状态的监控，可以通过MongoDB本身的mongosh命令行或MongoDB Compass进行访问（以及HDFS的http://localhost:9870/）

扩展功能

1. MongoDB自身的部署复制集功能可以实现错误容忍以及错误恢复（fault tolerance）
2. MongoDB可以在运行时通过`addShard`实现增加数据分片，实现了分布式数据库的热扩展
3. 我们可以使用mongosh完成更多配置（包括数据迁移等）

未来工作

由于时间所限，本项目还存在以下优化空间，可作为未来工作

1. 未使用内存数据库如Redis或维护内存中数据结构作为缓存，目前仅仅将数据存入MongoDB持久化道磁盘并读取，未来可以实现查询缓存以优化查询速度
2. 命令行交互界面尚未支持插入语句，插入词条需要借助mongosh进行交互，未来可以继续CLI的开发
3. 命令行界面尚未加入用户系统
4. 尚未测试整个系统的并发能力测试，而Golang在并发应用开发方面具有很大优势，未来也可以继续这方面的探索

结论

本项目达成了如下目标:

1. 达成了课程项目所提及的项目要求
2. 使用Docker Compose完成了分布式环境的部署并成功模拟MongoDB与HDFS集群而非单机上的伪分布式集群。
3. 对于MongoDB以及分布式数据库有了更深的认识和理解，通过实际的操作，理解分布式数据库中的数据分片与数据复制的实际应用。
4. 对于HDFS的应用有了些初步的认识，过去无数次听说过HDFS却未使用过，这次获得了实践的机会
5. 对于使用Golang搭建大型项目有了多经验，Golang作为近年来逐渐火热的语言，很适合分布式应用开发，结合Docker，在未来的工作中会受益无穷。

参考文献

1. Özsu, M. Tamer, and Patrick Valduriez. *Principles of distributed database systems*. Vol. 2. Englewood Cliffs: Prentice Hall, 1999.
2. MongoDB Manual - Replication https://docs.mongodb.com/manual/replication/
3. MongoDB Manual - Sharding https://docs.mongodb.com/manual/sharding/
4. HDFS Namenode and Datanode https://data-flair.training/forums/topic/explain-namenode-and-datanode-in-hdfs/
5. HDFS https://www.cnblogs.com/zhangyinhua/p/7657937.html
6. Hadoop配置 http://dblab.xmu.edu.cn/blog/2441-2/
7. Golang操作数据库 https://www.cnblogs.com/traditional/p/12534719.html

---

SYSTEM MANUAL

Installation

Test environments of the project

- macOS Catalina, Intel Core i7
- Golang 1.15
- Docker 20.10.11
    - Mongo 5.0.5 (Deployed using mongo:latest image)
    - Hadoop 3.3.1 (Deployed using ubuntu:latest image)

Configuration

- Docker configuration file please refer to `configs/docker-compose.yaml`
- MongoDB configuration file for each node please refer to `configs/mongo-docker`

- Using `mongos -f mongos.yaml` to innitiate the mongo router node
    - Other node can directly use the mongo image and initate using `--config`
    - Mongo router node can only start with `mongos`
        - The walkaround is to install MongoDB from a ubuntu image instead of using mongo image
- Configure the hdfs follow some classic tutorial

Some useful terminal commands

- *Show all containers ip address:*

  ```
  docker inspect --format='{{.Name}} - {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -aq)
  ```

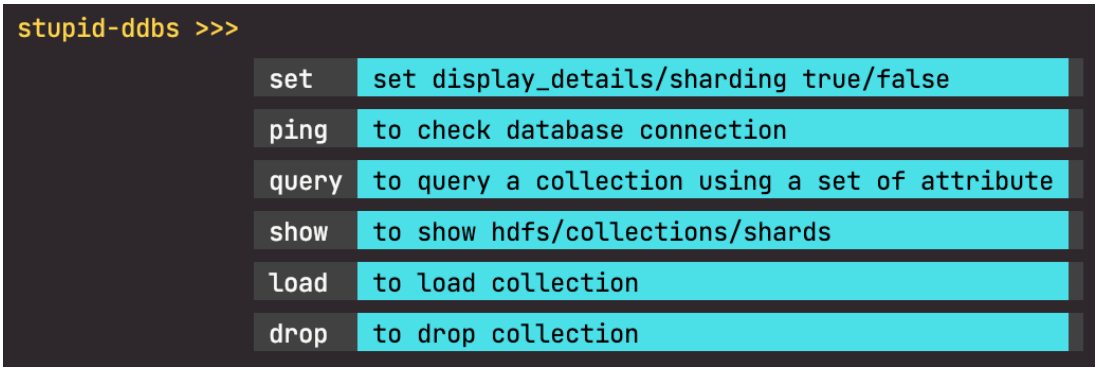
  > */config2 - 192.168.1.7*
  > */shard1_slave - 192.168.1.3*
  > */shard2_slave - 192.168.1.5*
  > */shard2_master - 192.168.1.4*
  > */config1 - 192.168.1.6*
  > */mongos1 - 192.168.1.8*
  > */hdfs - 192.168.1.9*
  > */shard1_master - 192.168.1.2*

MongoDB Cluster Setup

- entering shard1_master, shard2_master (using `mongosh`) and build two shards with replica mode
    - here s1m is the network alias name for shard1_master
    - ```
      config={
          _id:'rs1',members:[{
          _id:0,host:'s1m:27017',priority:2},{
          _id:1,host:'s1s:27017',priority:1}]}
      ```
    - `rs.initiate(config)`
- entering config1 and build config server shard
- entering router node and add shards
    - `sh.addShard("rs1/172.17.0.6:27017,172.17.0.5:27017")`
    - `sh.addShard("rs2/172.17.0.7:27017,172.17.0.3:27017")`
      ```

Operation

- Supported operation has been suggested in the CLI



- Some useful commands are listed here:
  - `query article aid le 99 category eq technology;`
  - query collections (user, read, article, beread, popular)
  - `show hdfs <path>`
  - `show collections`
  - `show shards`
  - `ping`
  - `set display_details true`: set if show picture and content from hdfs
  - `set sharding true`: shards initiate (zone assignment for each key should be configure in mongo router node)
  - `set logging true/false`